

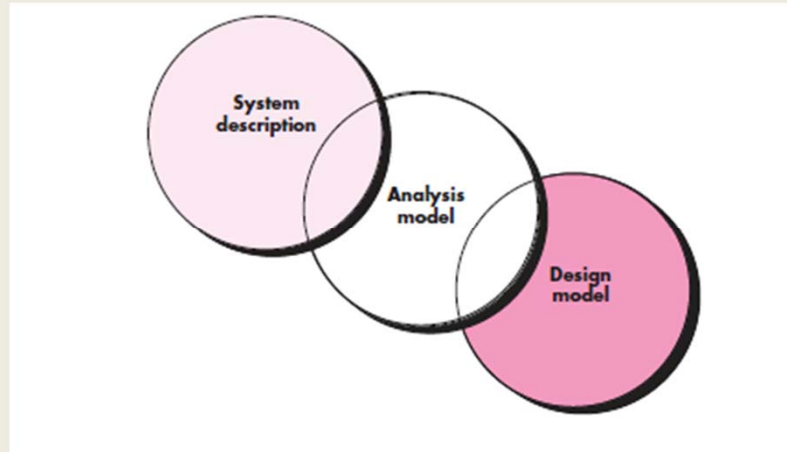
Requirements Modeling:  
Scenarios, information, and analysis classes



# 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.

## A Bridge



The requirements model as a bridge between the system description and the design model.

## 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

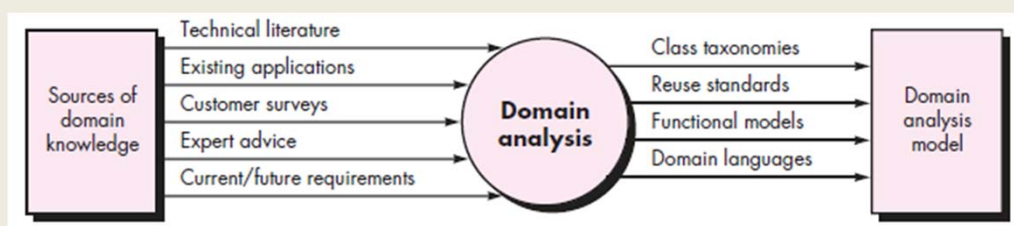
Software domain analysis is the identification, analysis, and specification of *common* requirements from *a specific application domain*, typically for *reuse* on multiple projects within that application domain . . .

*Object-oriented* domain analysis is the identification, analysis, and specification of *common, reusable capabilities within a specific application domain*, in terms of common objects, classes, subassemblies, and frameworks . . .

***Donald Firesmith***

# Domain Analysis

- 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.



domain analysis may be viewed as an umbrella activity for the software process. By this I mean that domain analysis is an ongoing software engineering activity that is not connected to any one software project.

This figure illustrates key inputs and outputs for the domain analysis process. Sources of domain knowledge are surveyed in an attempt to identify objects that can be reused across the domain.

## Requirements analysis approach

- 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
- Object-oriented analysis
  - Focuses on the definition of classes and the manner in which they collaborate with one another to effect customer requirements

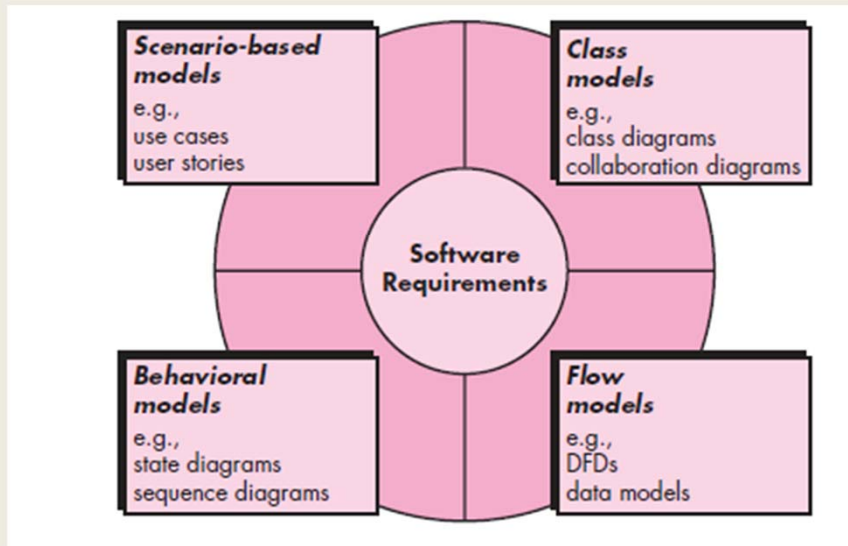
UML and the Unified Process are predominantly object oriented

# Building the Requirement Model

- Elements of the analysis model
  - Scenario-based elements
    - Functional—processing narratives for software functions
    - Use-case—descriptions of the interaction between an “actor” and the system. E.g. UML use-case diagram.
  - Class-based elements, E.g., UML class diagram.
    - Implied by scenarios.
    - Generalize classes, inheritance, and associations.
  - Behavioral elements
    - State diagram. E.g.: UML statechart.
  - Flow-oriented elements
    - Data flow diagram



## Elements of a requirement analysis model



Each element of the requirements model (Figure 6.3) presents the problem from a different point of view. 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 (some hierarchical) 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.

## Scenario-Based Modeling

- The key is to understand how end users want to **interact** with a system
  - UML results: **Use cases, activity diagrams, swimlane diagrams**
- Steps
  - Creating a Preliminary Use Case
  - Refining a Preliminary Use Case
  - Writing a Formal Use Case

Although the success of a computer-based system or product is measured in many ways, user satisfaction resides at the top of the list.

## 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
- Questions to consider
  - (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?*

a scenario that describes a “thread of usage” for a system

**actors** represent roles people or devices play as the system functions

**users** can play a number of different roles for a given scenario

## What to Write About?

- **Inception and elicitation**—provide you with the information you'll need to begin writing use cases.
- **Requirements gathering meetings, QFD, and other requirements engineering mechanisms** are used to
  - identify stakeholders
  - define the scope of the problem (p656)
  - specify overall operational goals
  - establish priorities
  - outline all known functional requirements, and
  - describe the things (objects) that will be manipulated by the system.
- 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.

## Creating 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**?

## Creating preliminary use case – SafeHome Example

- Identifies the following functions (an abbreviated list) that are performed by the **homeowner** actor
  - Select camera to view.
  - Request thumbnails from all cameras.
  - Display camera views in a PC window.
  - Control pan and zoom for a specific camera.
  - Selectively record camera output.
  - Replay camera output.
  - Access camera surveillance via the Internet.

## Creating preliminary use case – SafeHome Example

- Function: ACS-DCV
  - Access camera surveillance via the Internet –display camera views
- Use case: Access camera surveillance via the Internet—display camera views (ACS-DCV)
- Actor: homeowner

If I'm at a remote location, I can use any PC with appropriate browser software to log on to the *SafeHomeProducts* website. I enter my user ID and two levels of passwords and once I'm validated, I have access to all functionality for my installed *SafeHomesystem*. To access a specific camera view, I select "surveillance" from the major function buttons displayed. I then select "pick a camera" and the floor plan of the house is displayed. I then select the camera that I'm interested in. Alternatively, I can look at thumbnail snapshots from all cameras simultaneously by selecting "all cameras" as my viewing choice. Once I choose a camera, I select "view" and a one-frame-per-second view appears in a viewing window that is identified by the camera ID. If I want to switch cameras, I select "pick a camera" and the original viewing window disappears and the floor plan of the house is displayed again. I then select the camera that I'm interested in. A new viewing window appears.



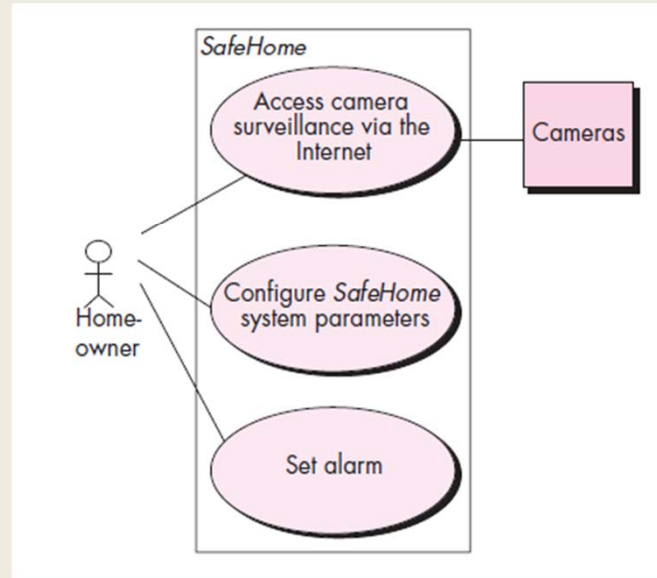
## Creating preliminary use case – SafeHome Example

- Scenario (primary)
  - 1. The homeowner logs onto the *SafeHomeProducts* website.
  - 2. The homeowner enters his or her user ID.
  - 3. The homeowner enters two passwords (each at least eight characters in length).
  - 4. The system displays all major function buttons.
  - 5. The homeowner selects the “surveillance” from the major function buttons.
  - 6. The homeowner selects “pick a camera.”
  - 7. The system displays the floor plan of the house.
  - 8. The homeowner selects a camera icon from the floor plan.
  - 9. The homeowner selects the “view” button.
  - 10. The system displays a viewing window that is identified by the camera ID.
  - 11. The system displays video output within the viewing window at one frame per second.

this sequential presentation does not consider any alternative interactions (the narrative is more free-flowing and did represent a few alternatives).

Use cases of this type are sometimes referred to as *primary scenarios*

## Creating preliminary use case – SafeHome Example



Preliminary use-case diagram for the *SafeHome* system. Each use case is represented by an oval. Only the

**ACS-DCV** use case has been discussed in this section.

In many cases, there is no need to create a graphical representation of a usage

scenario. However, diagrammatic representation can facilitate understanding, particularly

when the scenario is complex.

## Refining preliminary use case – SafeHome Example

- each step in the primary scenario is evaluated by asking the following questions
  - 1) *Can the actor take some other action at this point?*
  - 2) *Is it possible that the actor will encounter some error condition at this point? If so, what might it be?*
  - 3) *Is it possible that the actor will encounter some other behavior at this point (e.g. behavior that is invoked by some event outside the actor's control)? If so, what might it be?*
- Leads to a set of *secondary scenarios*

Answers to these questions result in the creation of a set of *secondary scenarios* that are part of the original use case but represent alternative behavior.

## Refining preliminary use case – SafeHome Example

- Consider steps 6-7:
  - 6. The homeowner selects “pick a camera.”
  - 7. The system displays the floor plan of the house.
- 1) Can the actor take some other action at this point?
  - The answer is “yes.” the actor may choose to view thumbnail snapshots of all cameras simultaneously.
  - one secondary scenario might be “View thumbnail snapshots for all cameras.”

## Refining preliminary use case – SafeHome Example

- 2) *Is it possible that the actor will encounter some error condition at this point?*
  - Any number of error conditions can occur as a computer-based system operates.
  - The answer is “yes.”
  - A floor plan with camera icons may have never been configured.
  - selecting “pick a camera” results in an error condition: “No floor plan configured for this house.” a secondary scenario


## Refining preliminary use case – SafeHome Example

- 3) *Is it possible that the actor will encounter some other behavior at this point?*
  - The answer is “yes.”
  - the system may encounter an alarm condition.
  - This would result in the system displaying a special alarm notification (type, location, system action) and providing the actor with a number of options relevant to the nature of the alarm.
  - It will not become part of the **ACS-DCV** use case.
  - Rather, [a separate use case](#)—**Alarm condition encountered**—would be developed and referenced from other use cases as required.

Because this secondary scenario *can occur at any time for virtually all interactions*, it will not become part of the **ACS-DCV** use case.

Each of the situations described in the preceding situations is characterized as a use-case exception. An *exception* describes a situation (either a failure condition or an alternative chosen by the actor) that causes the system to exhibit somewhat different behavior.

# Writing a formal use case

 Use case: Access camera surveillance via the Internet—display camera views (ACS-DCV)	
<b>Iteration:</b>	2, last modification: January 14 by V. Raman.
<b>Primary actor:</b>	Homeowner.
<b>Goal in context:</b>	To view output of camera placed throughout the house from any remote location via the Internet.
<b>Preconditions:</b>	System must be fully configured; appropriate user ID and passwords must be obtained.
<b>Trigger:</b>	The homeowner decides to take a look inside the house while away.
<b>Scenario:</b>	<ol style="list-style-type: none"> <li>1. The homeowner logs onto the SafeHome Products website.</li> <li>2. The homeowner enters his or her user ID.</li> <li>3. The homeowner enters two passwords (each at least eight characters in length).</li> <li>4. The system displays all major function buttons.</li> <li>5. The homeowner selects the "surveillance" from the major function buttons.</li> <li>6. The homeowner selects "pick a camera."</li> <li>7. The system displays the floor plan of the house.</li> <li>8. The homeowner selects a camera icon from the floor plan.</li> <li>9. The homeowner selects the "view" button.</li> <li>10. The system displays a viewing window that is identified by the camera ID.</li> <li>11. The system displays video output within the viewing window at one frame per second.</li> </ol>
<b>Exceptions:</b>	<ol style="list-style-type: none"> <li>1. ID or passwords are incorrect or not recognized—see use case <b>Validate ID and passwords</b>.</li> <li>2. Surveillance function not configured for this system—system displays appropriate error message; see use case <b>Configure surveillance function</b>.</li> <li>3. Homeowner selects "View thumbnail snapshots for all camera"—see use case <b>View thumbnail snapshots for all cameras</b>.</li> <li>4. A floor plan is not available or has not been configured—display appropriate error message and see use case <b>Configure floor plan</b>.</li> <li>5. An alarm condition is encountered—see use case <b>Alarm condition encountered</b>.</li> </ol>
<b>Priority:</b>	Moderate priority, to be implemented after basic functions.
<b>When available:</b>	Third increment.
<b>Frequency of use:</b>	Moderate frequency.
<b>Channel to actor:</b>	Via PC-based browser and Internet connection.
<b>Secondary actors:</b>	System administrator, cameras.
<b>Channels to secondary actors:</b>	<ol style="list-style-type: none"> <li>1. System administrator: PC-based system.</li> <li>2. Cameras: wireless connectivity.</li> </ol>
<b>Open issues:</b>	<ol style="list-style-type: none"> <li>1. What mechanisms protect unauthorized use of this capability by employees of SafeHome Products?</li> <li>2. Is security sufficient? Hacking into this feature would represent a major invasion of privacy.</li> <li>3. Will system response via the Internet be acceptable given the bandwidth required for camera views?</li> <li>4. Will we develop a capability to provide video at a higher frames-per-second rate when high-bandwidth connections are available?</li> </ol>

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

The *goal in context* identifies the overall scope of the use case.

The *precondition* describes what is known to be true before the use case is initiated.

The *trigger* identifies the event or condition that "gets the use case started"

The *scenario* lists the specific actions that are required by the actor and the appropriate system responses.

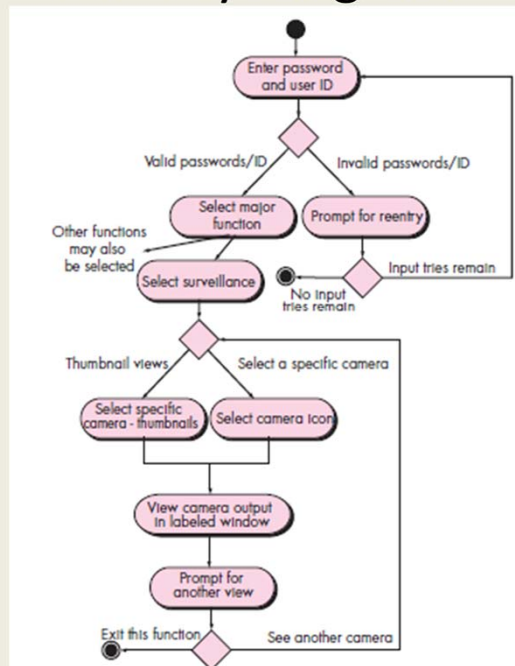
*Exceptions* identify the situations uncovered as the preliminary use case is refined

## Use case: discussion

- Benefit
  - If developed properly, the use case can provide substantial benefit as a modeling tool.
- Limitation
  - If the description is unclear, the use case *can be misleading or ambiguous*.
  - A use case focuses on functional and behavioral requirements and is generally *inappropriate for nonfunctional requirements*.
  - For situations in which the requirements model must have significant detail and precision (e.g., safety critical systems), a use case *may not be sufficient*.



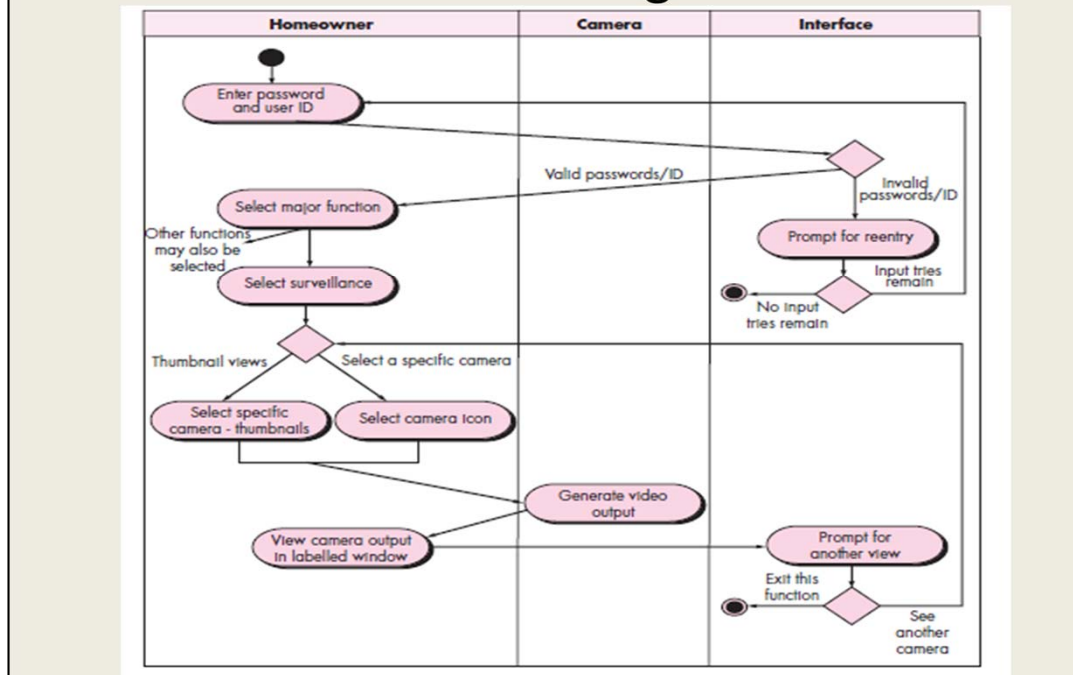
## Activity Diagram



Supplements the use-case by providing a diagrammatic representation of procedural flow

There are many requirements modeling situations in which a text-based model—even one as simple as a use case—may not impart information in a clear and concise manner.

## Swimlane diagram



Extension of activity diagram.

Swimlane diagram Allows the modeler to represent the flow of activities described by the use-case and at the same time indicate which actor (if there are multiple actors involved in a specific use-case) or analysis class has responsibility for the action described by an activity rectangle

## Summary

- The role of requirement analysis model in software process
- Scenario-based models
  - Use case: a narrative or template-driven description of an interaction b/w actor and the software
    - Primary modeling element for scenario-based model
- UML models for representing scenarios
  - Activity diagram: extended use case diagram
  - Swimlane diagram: how the processing flow is allocated to various actors or classes

Release Milestone 2