Requirement Engineering

- The process of collecting the software requirement from the client then understand, evaluate and document it is called as requirement engineering.
- **Requirements engineering** (RE) is the process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
- Requirement engineering constructs a bridge for design and construction.

Requirement Engineering Tasks

- **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

- **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.

- **Elaboration**

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

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

Requirement Engineering Tasks

- **Validation**
- The work product is built as an output of the requirement engineering and that is accessed for the quality through a validation step.
- **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.

Establishing the groundwork

Requirements engineering is simply a matter of conducting meaningful conversations with colleagues who are well-known members of the team

- **Identifying Stakeholders**
- **Recognizing Multiple Viewpoints**
- **Working toward Collaboration**
- **Asking the First Questions**
- **Non Functional requirements**
- **Traceability**

Eliciting Requirements

- **Collaborative Requirements Gathering:**
  - Guideslines to be followed
  - *goal*
  - lists to be prepared
- **QFD Purpose,**
  - Types of QFDs
  - Customer voice table
  - Purpose of CVT
- **Usage scenarios**
- **Work products produced**

Eliciting Requirements

- **Requirements elicitation** (also called *requirements gathering*) **combines elements of problem solving, elaboration, negotiation, and specification**.

Collaborative Requirements Gathering:

Guideslines to be followed

- **Meetings are conducted and attended** by both software engineers and customers
- **Rules** for preparation and participation are **established**
- An **agenda** is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas.
- A "**facilitator**" (can be a customer, a developer, or an outsider) controls the meeting
- A "**definition mechanism**" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used

- The ***goal*** is
  - **to identify the problem**
  - **propose elements of the solution**
  - **negotiate different approaches, and**
  - **specify a preliminary set of solution requirements**

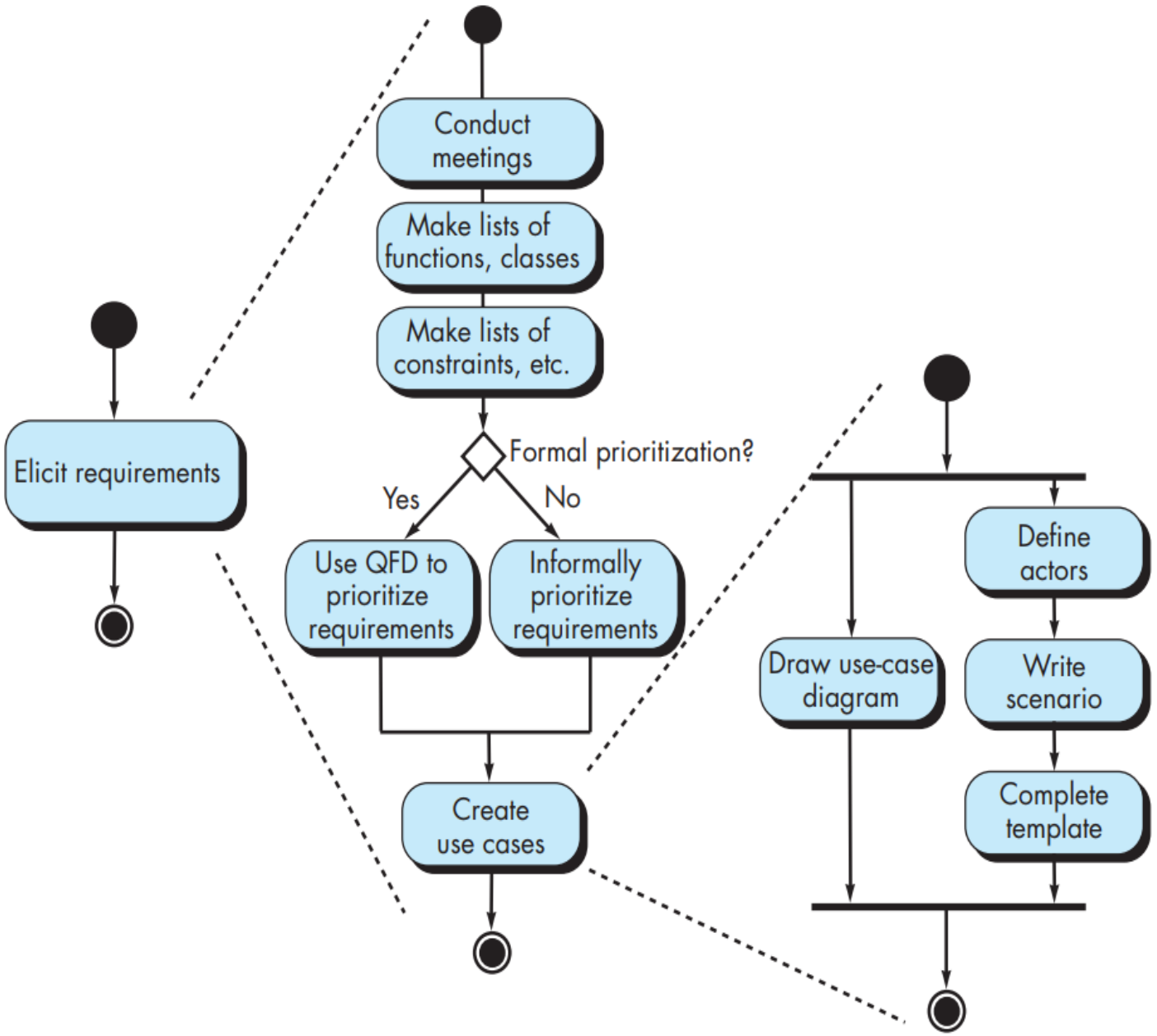
Eliciting Requirements

- During inception basic questions and answers establish the scope of the problem and the overall perception of a solution.
- Out of these **initial meetings, the developer and customers write a one- or two-page "product request."**
- A meeting place, time, and date are selected; a facilitator is chosen; and attendees from the software team and other stakeholder organizations are **invited to participate.**
- The **product request is distributed to all attendees** before the meeting date.
- **E.g.: Safe Home Security function**
- Marketing person writes the following narrative:
  - **Our research indicates that the market for home management systems is growing at a rate of 40 per cent per year. The first *Safe Home* function we bring to market should be the home security function. Most people are familiar with "alarm systems" so this would be an easy sell.**
  - **The home security function would protect against and/or recognize a variety of undesirable "situations" such as illegal entry, fire, flooding, carbon monoxide levels, and others.**

Eliciting Requirements

- While reviewing the product request in the days before the meeting,
  - **each attendee is asked to make a list of objects that are part of the environment that surrounds the system**,
  - **other objects** that are **to be produced** by the system, and
  - objects that are **used** by the system to **perform its functions**.
  - Each attendee is asked to **make another list of services** (processes or functions) **that manipulate or interact with the objects**.
  - **Lists of constraints** (e.g., cost, size, business rules) and **performance criteria** (e.g., speed, accuracy) are also developed.
- The attendees are informed that the lists are not expected to be exhaustive but are expected to reflect each person's perception of the system.
  - **Objects described for *Safe Home* might include the control panel, smoke detectors, window and door sensors, motion detectors, an alarm, an event (a sensor has been activated), a display, a PC, telephone numbers, a telephone call, and so on. The list of services might include *configuring* the system, *setting* the alarm, *monitoring* the sensors, *dialling* the phone, *programming* the control panel, and *reading* the display(note that services act on objects).**

UML activity diagrams for eliciting requirements



Quality Function Deployment

- Function deployment determines the “value” (as perceived by the customer) of each function required of the system
- **Information deployment** identifies data objects and events
- **Task deployment** examines the behavior of the system
- **Value analysis** determines the relative priority of requirements

Quality Function Deployment

- *Quality function deployment* (QFD) is a **quality management technique that translates the needs of the customer into technical requirements for software.**
- QFD "**concentrates on maximizing customer satisfaction from the software engineering process**".
- To accomplish this, QFD **emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the engineering process.**

Quality Function Deployment

- **QFD identifies three types of requirements.**
  - **Normal requirements.** The objectives and goals that are stated for a product or system during meetings with the customer. If these requirements are present, the **customer is satisfied**.
    - **Examples of normal requirements might be requested types of graphical displays, specific system functions, and defined levels of performance.**
  - **Expected requirements.** These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them. **Their absence will be a cause for significant dissatisfaction.**
    - **Examples of expected requirements are: ease of human/machine interaction, overall operational correctness and reliability, and ease of software installation.**
  - **Exciting requirements.** These features go **beyond the customer's expectations** and prove to be very satisfying when present.
    - **For example, software for a new mobile phone comes with standard features, but is coupled with a set of unexpected capabilities (e.g., multi touch screen, visual voice mail) that delight every user of the product.**

Quality Function Deployment

- QFD uses **customer interviews and observation, surveys, and examination of historical data** (e.g., **problem reports**) as **raw data** for the ***requirements gathering activity***.
- These **data** are then **translated into a table of requirements**—called the ***customer voice table***—that is reviewed with the customer and other stakeholders.
- A variety of **diagrams, matrices, and evaluation methods are then used** to **extract expected requirements** and to attempt to derive exciting requirements

Usage Scenarios

- As requirements are gathered, an **overall vision of system functions and features begins to materialize.**
- These are used to understand **how these functions and features will be used** by different classes of end users.
- To accomplish this, **developers and users can create a set of scenarios that identify a thread of usage for the system to be constructed.**
- The scenarios, often called ***use cases***, provide a description of **how the system will be used**.

Elicitation Work Products

- The **work products produced** as a consequence of requirements elicitation will **vary depending on the size of the system or product to be built**.
- For most systems, the work products include
  - **a statement of need and feasibility.**
  - **a bounded statement of scope for the system or product.**
  - **a list of customers, users, and other stakeholders who participated in requirements elicitation**
  - **a description of the system's technical environment.**
  - **a list of requirements (preferably organized by function) and the domain constraints that apply to each.**
  - **a set of usage scenarios that provide insight into the use of the system or product under different operating conditions.**
  - **any prototypes developed to better define requirements.**
- Each of these **work products is reviewed** by all people who have participated in requirements elicitation.

Developing Usecases

USE CASES

It is helpful to create scenarios that identify a thread of usage for the system to be constructed.

These scenarios are often called use cases.

Use cases tell a story about how an end user interacts with the system under a specific set of circumstances.

This story may be:

- ***narrative text***
- ***an outline of tasks or interactions***
- ***a template-based description***
- ***diagrammatic representation***

It depicts software from an end-user point of view.

Definition of USECASE:

"A use case captures a contract.. [that] describes the system's behavior under various conditions as the system responds to a request from one of its stakeholders."

Developing Usecases

- **Use case** is to define the **set of "actors"** that will be involved in the story.
- **Actors** are the different people (or devices) that use the system or product within the **context of the function or behaviour** that is to be described. Actors represent the **roles** that people (or devices) play as the system operates.
- An **actor** is anything that **communicates** with the system or product and that is **external in the system itself**.
- Every actor has **one or more goals** when using the system.
- **Actor and end user** are not necessarily the **same** thing.
- A typical **user** may play a number of **different roles** when using a system, whereas an actor represents a **class of external entities** (often, but not always, people) that **play just one role in the context of the use case**.
  - Eg: **consider a machine operator (a user) who interacts with the control computer for a manufacturing cell that contains a number of robots and numerically controlled machines.**

Developing Use cases

- After careful review of requirements, the software for the control computer requires **four different modes** (roles) for interaction **programming mode, test mode, monitoring mode, and trouble shooting mode**.
- Four actors can be defined: **programmer, tester, monitor, and trouble shooter.**
- In some cases, **the machine operator can play all these roles**.
- In others, **different people may play the role of each actor**.
- In the **first iteration *primary actors*** are identified
- **After first iteration *secondary actors*** are identified if necessary.
- ***"Primary actors"*** **interact to achieve required system function** and **derive** the intended **benefit** from the system.
- They **work directly and frequently** with the software.
- ***"Secondary actors"*** **support the system** so that primary actors can do their work.

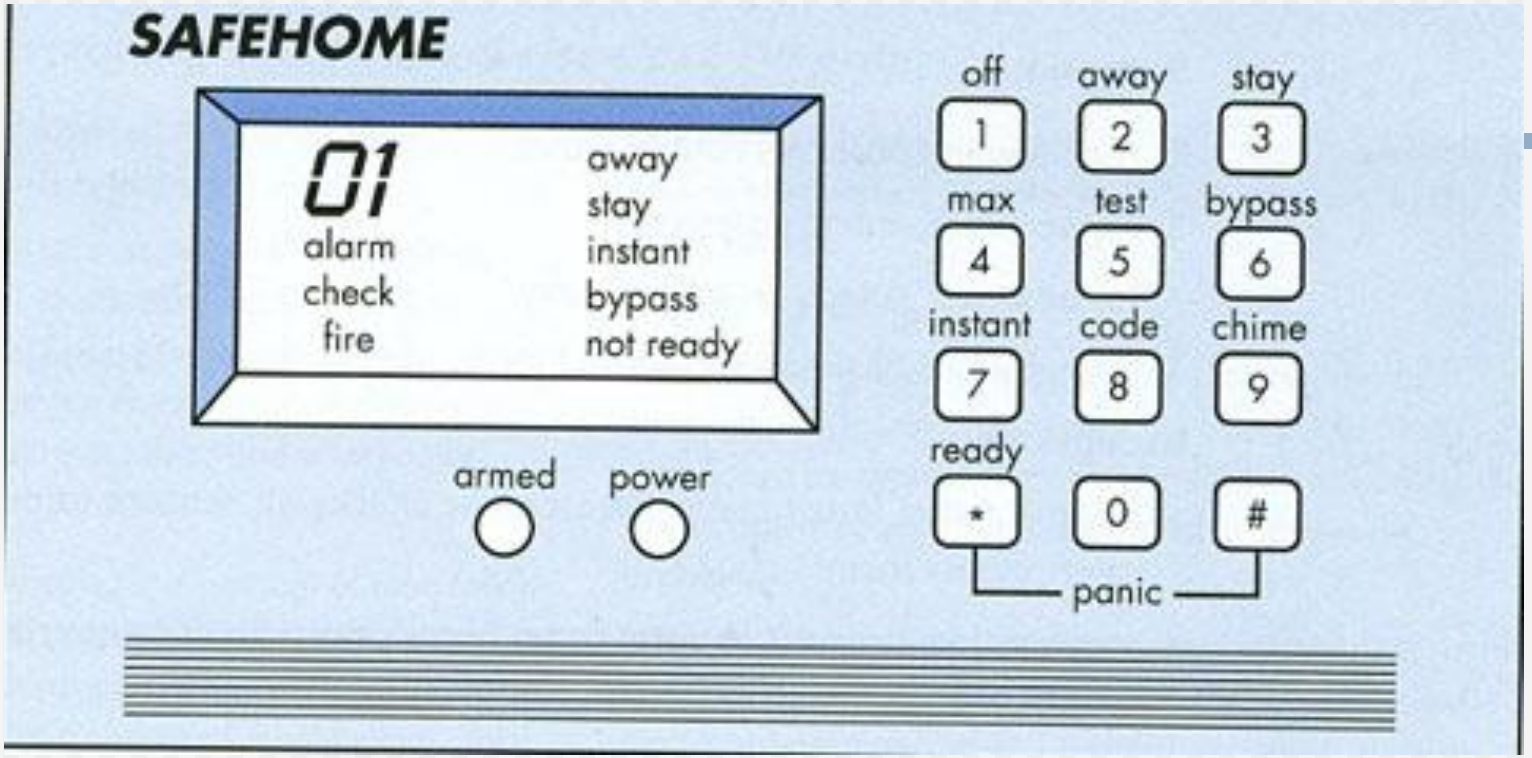
Use-Cases

- 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?

DEVELOPING USE CASES

Think about a home security system.



DEVELOPING USE CASES

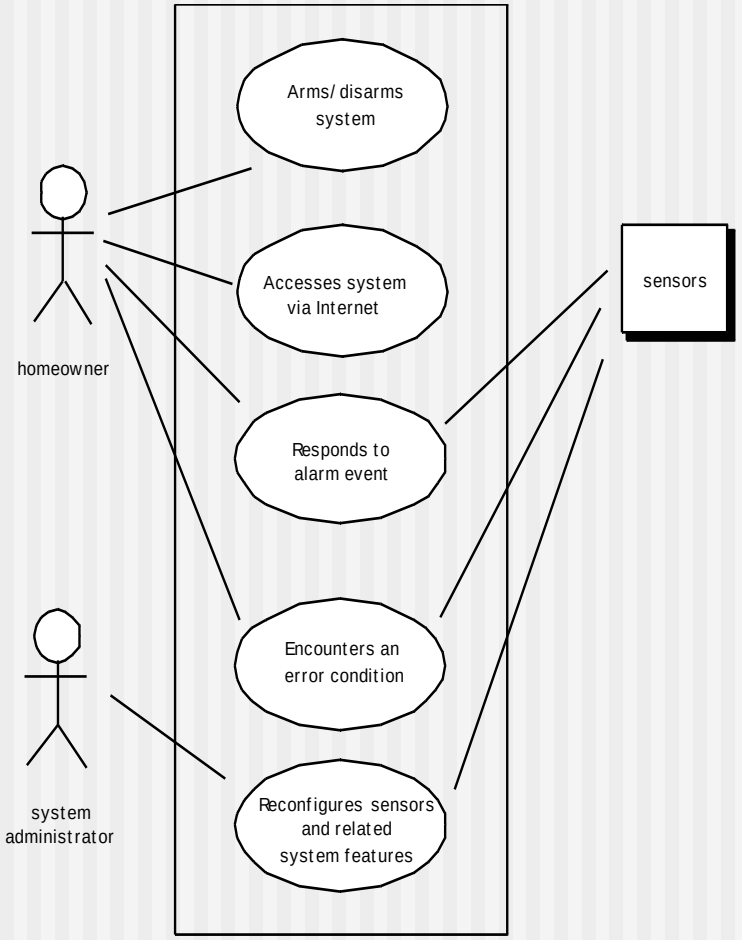
Think about a home security system.

Three actors exist: homeowner/configuration manager, sensors, and the monitoring subsystem.

Let's look at the homeowner. The homeowner interacts with the home security function in a number of different ways using either the alarm control panel or a PC:

- **Enters a password to allow *all other interactions*.**
- **Inquires about the *status* of a *security zone.***
- **Inquiries about the *status of a sensor*.**
- **Presses the *panic button* in an emergency.**
- **Activates/deactivates the security system.**

Use-Case Diagram

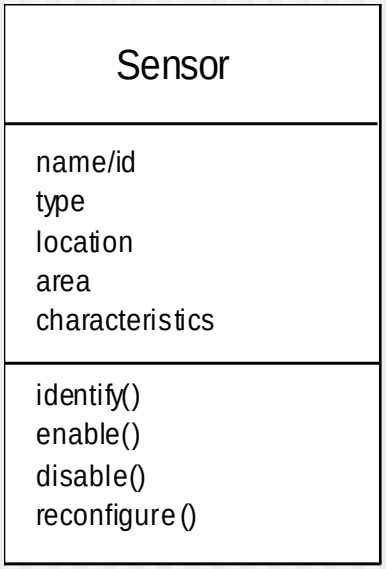


Building the Analysis Model

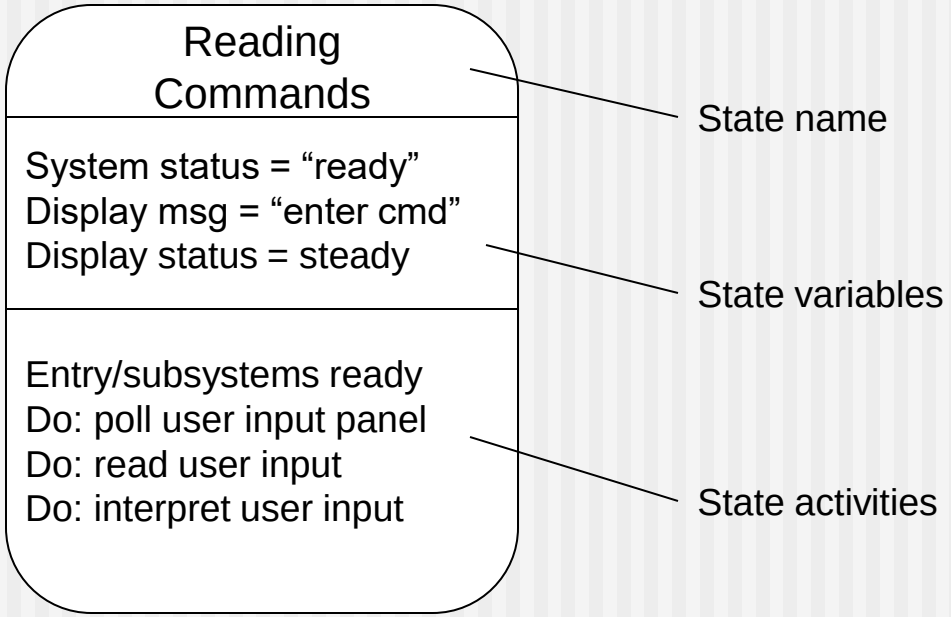
- Elements of the analysis model
  - Scenario-based elements
    - Functional—processing narratives for software functions
    - Use-case—descriptions of the interaction between an "actor" and the system
  - Class-based elements
    - Implied by scenarios
  - Behavioral elements
    - State diagram
  - Flow-oriented elements
    - Data flow diagram

Class Diagram

From the *SafeHome* system …



State Diagram



Analysis Patterns

Pattern name: A descriptor that captures the essence of the pattern.

Intent: Describes what the pattern accomplishes or represents

Motivation: A scenario that illustrates how the pattern can be used to address the problem.

Forces and context: A description of external issues (forces) that can affect how the pattern is used and also the external issues that will be resolved when the pattern is applied.

Solution: A description of how the pattern is applied to solve the problem with an emphasis on structural and behavioral issues.

Consequences: Addresses what happens when the pattern is applied and what trade-offs exist during its application.

Design: Discusses how the analysis pattern can be achieved through the use of known design patterns.

Known uses: Examples of uses within actual systems.

Related patterns: On e or more analysis patterns that are related to the named pattern because (1) it is commonly used with the named pattern; (2) it is structurally similar to the named pattern; (3) it is a variation of the named pattern.

Negotiating Requirements

- Identification of system key stakeholders
- Determination of stakeholders' "win conditions"
- Negotiate to reconcile stakeholders' win conditions into "win-win" result for all stakeholders (including developers)

The goal of requirement negotiation is to produce a win-win result before proceeding to subsequent software engineering activities.

Scenario-Based Modeling

- Computer-based system or product is measured in many ways, ***user satisfaction resides at the top of the list.***
- Software team will be better able to properly characterize requirements and build meaningful analysis and design models, if how end users (and other actors) want to interact with a system,
- Hence, requirements modelling with UML begins with the creation of scenarios in the form of use cases, activity diagrams, and swimlane diagrams.

Scenario-Based Modeling

Alistair Cockburn characterizes a use case as a "contract for behaviour". The "contract" defines the way in which an actor uses a computer-based system to accomplish some goal.

"[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, QFD, and other requirements engineering mechanisms are used to
  - **identify stakeholders**
  - **define the scope of the problem**
  - **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.

What to Write About?

- The *SafeHome* home surveillance function (subsystem) 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.

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.

How Much to Write About?

- To illustrate, consider the function *access camera surveillance via the Internet— display camera views* **(ACS-DCV).** The stakeholder who takes on the role of the **homeowner** actor might write the following narrative:
- **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 *SafeHome Products* 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 *SafeHome* system. 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.**

How Much to Write About?

- **A variation of a narrative use case presents the interaction as an ordered sequence of user actions. Each action is represented as a declarative sentence. Revisiting the ACS-DCV function, you would write:**
- **Use case: Access camera surveillance via the Internet—display camera views (ACS-DCV)**
- **Actor: homeowner**

1. The homeowner logs onto the *SafeHome Products* 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.

- **It is important to note that 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***

How Much to Write About?

- Refining a Preliminary Use Case:
- **A description of alternative interactions is essential for a complete understanding of the function that is being described by a use case. Therefore, each step in the primary scenario is evaluated by asking the following questions**
  - ***Can the actor take some other action at this point?***
  - ***Is it possible that the actor will encounter some error condition at this point?* If so, what might it be?**
  - ***Is it possible that the actor will encounter some other behaviour at this point (e.g.,behaviour that is invoked by some event outside the actor's control)?* If so, what might it be?**
- **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 behaviour. For example, consider steps 6 and 7 in the primary scenario presented earlier:**

6. **The homeowner selects "pick a camera."**
7. **The system displays the floor plan of the house.**

- ***Can the actor take some other action at this point?***
- **The answer is "yes." Referring to the free-flowing narrative, the actor may choose to view thumbnail snapshots of all cameras simultaneously. Hence, one secondary scenario might be "View thumbnail snapshots for all cameras."**

How Much to Write About?

- ***Is it possible that the actor will encounter some error condition at this point?***
- ***Ans:*** We consider only error conditions that are likely as a direct result of the action described in step 6 or step 7. Again the answer to the question is ***"yes."***

  A floor plan with camera icons may have never been configured**. Hence, selecting "pick a camera" results in an error condition: "No floor plan configured for this house." This error condition becomes a secondary scenario.**

- ***Is it possible that the actor will encounter some other behaviour at this point?***
- **Ans:** Again the answer to the question is ***"yes."*** As steps 6 and 7 occur, ***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**. Because this secondary scenario can occur at any time for virtually all interactions, 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.

How Much to Write About?

- Each of the situations described in the preceding paragraphs 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.
- **The following issues should also be explored:**
- ***Are there cases in which some "validation function" occurs during this use case?***
  - **This implies that validation function is invoked and a potential error condition might occur.**
- ***Are there cases in which a supporting function (or actor) will fail to respond appropriately****?*
  - **For example, a user action awaits a response but the function that is to respond times out.**
- ***Can poor system performance result in unexpected or improper user actions?***
  - **For example, a Web-based interface responds too slowly, resulting in a user making multiple selects on a processing button. These selects queue inappropriately and ultimately generate an error condition.**

Writing a Formal Use Case:

SafeHome

Use Case Template for Surveillance

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

Iteration: 2, last modification: January 14 by V. Raman.

Primary actor: Homeowner.

Goal in context: To view output of camera placed throughout the house from any remote location via the Internet.

Preconditions: System must be fully configured; appropriate user ID and passwords must be obtained.

Trigger: The homeowner decides to take a look inside the house while away.

Scenario:

1. The homeowner logs onto the *SafeHome Products* 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.

Exceptions:

1. ID or passwords are incorrect or not recognized—see use case **Validate ID and passwords.**
2. Surveillance function not configured for this system—system displays appropriate error message; see use case **Configure surveillance function.**
3. Homeowner selects "View thumbnail snapshots for all camera"—see use case **View thumbnail snapshots for all cameras.**
4. A floor plan is not available or has not been configured—display appropriate error message and see use case **Configure floor plan.**
5. An alarm condition is encountered—see use case **Alarm condition encountered.**

Priority: Moderate priority, to be implemented after basic functions.

When available: Third increment.

Frequency of use: Moderate frequency.

Channel to actor: Via PC-based browser and Internet connection.

Secondary actors: System administrator, cameras.

Channels to secondary actors:

1. System administrator: PC-based system.
2. Cameras: wireless connectivity.

Open issues:

1. What mechanisms protect unauthorized use of this capability by employees of *SafeHome Products*?
2. Is security sufficient? Hacking into this feature would represent a major invasion of privacy.
3. Will system response via the Internet be acceptable given the bandwidth required for camera views?
4. Will we develop a capability to provide video at a higher frames-per-second rate when high-bandwidth connections are available?

Writing a Formal Use Case:

- The **ACS-DCV** use case shown in the sidebar follows a typical outline for formal use cases. The *goal in context* identifies the overall scope of the use case. The *recondition* 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 Diagram