# Requirements Elicitation

Use Case & Scenarios

# First step in identifying the Requirements: System identification

- Two questions need to be answered:
  1. How can we identify the purpose of a system?
  2. What is inside, what is outside the system?
- These two questions are answered during requirements elicitation and analysis
- Requirements elicitation:
  - Definition of the system in terms understood by the customer ("Requirements specification")
- Analysis:
  - Definition of the system in terms understood by the developer (Technical specification, "Analysis model")
- Requirements Process: Contains the activities Requirements Elicitation and Analysis.

# Types of Requirements

- Functional requirements
  - Describe the interactions between the system and its environment independent from the implementation

    "An operator must be able to define a new game."

- Nonfunctional requirements
  - Aspects not directly related to functional behavior.

    "The response time must be less than 1 second"

- Constraints
  - Imposed by the client or the environment
    - "The implementation language must be Java "
  - Called "Pseudo requirements" in the text book.

# Functional vs. Nonfunctional Requirements

## Functional Requirements

- Describe user tasks that the system needs to support
- Phrased as actions
  - "Advertise a new league"
  - "Schedule tournament"
  - "Notify an interest group"

## Nonfunctional Requirements

- Describe properties of the system or the domain
- Phrased as constraints or negative assertions
  - "All user inputs should be acknowledged within 1 second"
  - "A system crash should not result in data loss".

# Types of Nonfunctional Requirements

## Quality requirements

- Usability - the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.
- Reliability - the ability of a system or component to perform its required functions under stated conditions for a specified period of time. Recently, this category is often replaced by **dependability**, which is the property of a computer system such that reliance can justifiably be placed on the service it delivers. Dependability includes **robustness** and **safety**.
- Performance requirements - are concerned with quantifiable attributes of the system, such as: **Response time**, **Throughput**, **Availability**

# Types of Nonfunctional Requirements_2

## Quality requirements

- Supportability requirements - are concerned with the ease of changes to the system after deployment, including for example, **adaptability** (the ability to change the system to deal with additional application domain concepts), **maintainability** (the ability to change the system to deal with new technology or to fix defects), and internationalization.

## Constraints (Pseudo requirements)

- Implementation requirements - are constraints on the implementation of the system, including the use of specific tools, programming languages, or hardware platforms.

# Types of Nonfunctional Requirements_3

## Constraints (Pseudo requirements)

- Implementation requirements - are constraints on the implementation of the system, including the use of specific tools, programming languages, or hardware platforms.
- Interface requirements - are constraints imposed by external systems, including legacy systems and interchange formats.
- Operations requirements - are constraints on the administration and management of the system in the operational setting.
- Packaging requirements - are constraints on the actual delivery of the system (e.g., constraints on the installation media for setting up the software).

# Types of Nonfunctional Requirements_4

| | |
|---|---|
| **Constraints (Pseudo requirements)** | Legal requirements - are concerned with licensing, regulation, and certification issues. An example of a legal requirement is that software developed for the U.S. federal government must comply with Section 508 of the Rehabilitation Act of 1973, requiring that government information systems must be accessible to people with disabilities. |

# Nonfunctional Requirements (Quality): Examples

- "Spectators must be able to watch a match without prior registration and without prior knowledge of the match."

  → *Usability Requirement*

- "The system must support 10 parallel tournaments"

  → *Performance Requirement*

- "The operator must be able to add new games without modifications to the existing system."

# What should not be in the Requirements?

- System structure, implementation technology
- Development methodology
- Development environment
- Implementation language
- Reusability

- It is desirable that none of these above are constrained by the client. Fight for it!

# Requirements Validation

Requirements validation is a quality assurance step, usually performed after requirements elicitation or after analysis

- Correctness:
  - The requirements represent the client's view
- Completeness:
  - All possible scenarios, in which the system can be used, are described
- Consistency:
  - There are no requirements that contradict each other.

# Requirements Validation (2)

- Clarity:
  - Requirements can only be interpreted in one way
- Realism:
  - Requirements can be implemented and delivered
- Traceability:
  - Each system behavior can be traced to a set of functional requirements

- Problems with requirements validation:
  - Requirements change quickly during requirements elicitation
  - Inconsistencies are easily added with each change
  - Tool support is needed!

# We can specify Requirements for “Requirements Management”

- Functional requirements:
  - Store the requirements in a shared repository
  - Provide multi-user access to the requirements
  - Automatically create a specification document from the requirements
  - Allow change management of the requirements
  - Provide traceability of the requirements throughout the artifacts of the system.

# Different Types of Requirements Elicitation

- Greenfield Engineering
  - Development starts from scratch, no prior system exists, requirements come from end users and clients
  - Triggered by user needs
- Re-engineering
  - Re-design and/or re-implementation of an existing system using newer technology
  - Triggered by technology enabler
- Interface Engineering
  - Provision of existing services in a new environment
  - Triggered by technology enabler or new market needs

# Prioritizing requirements

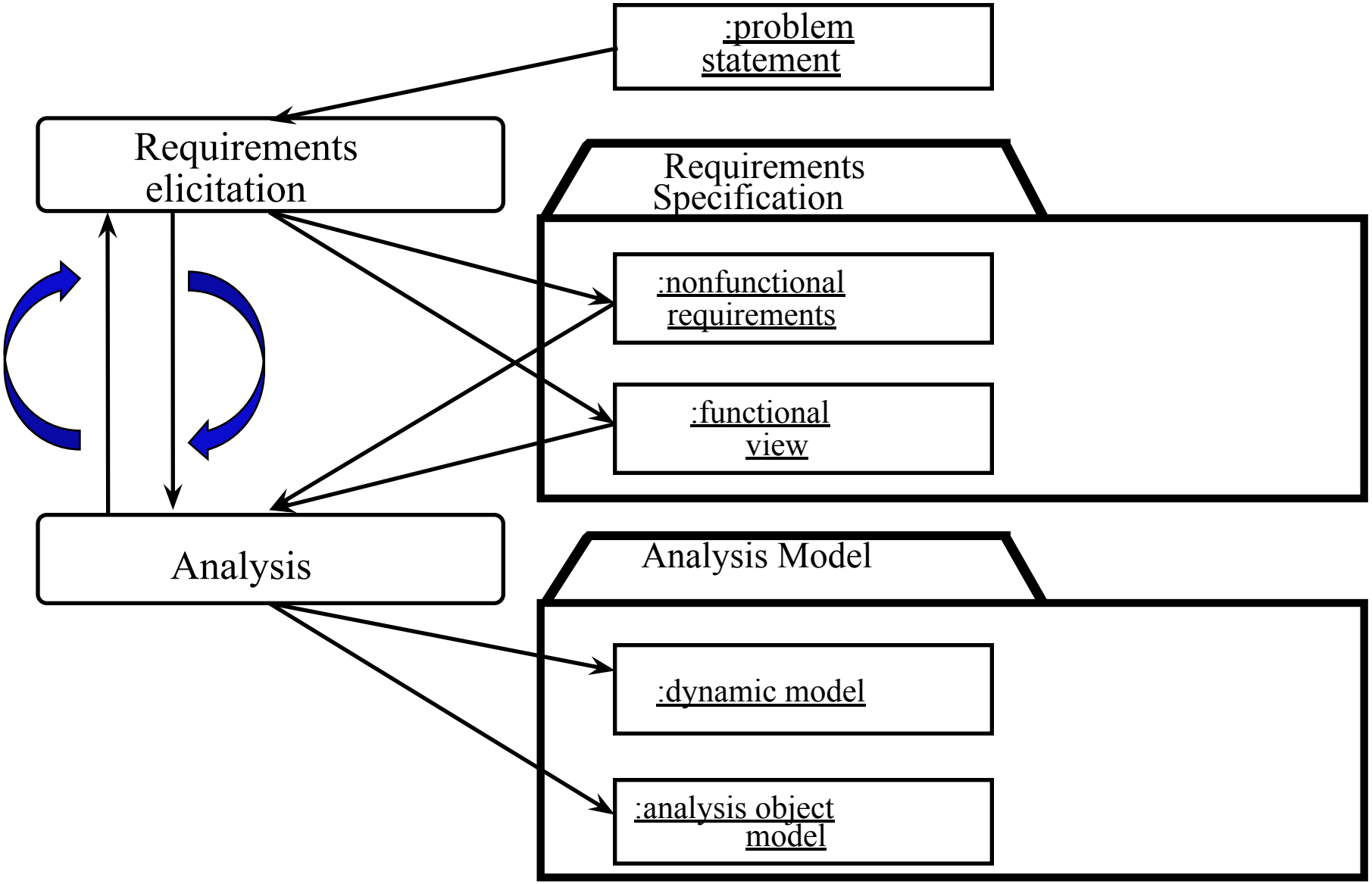
- High priority
  - Addressed during analysis, design, and implementation
  - A high-priority feature must be demonstrated
- Medium priority
  - Addressed during analysis and design
  - Usually demonstrated in the second iteration
- Low priority
  - Addressed only during analysis
  - Illustrates how the system is going to be used in the future with not yet available technology

# Requirements Specification vs Analysis Model

Both focus on the requirements from the user's view of the system

- The requirements specification uses natural language (derived from the problem statement)
- The analysis model uses a formal or semi-formal notation (we use UML)

# Requirements Process



# Techniques to elicit Requirements

- Bridging the gap between end user and developer:
  - **Questionnaires:** Asking the end user a list of pre-selected questions
  - **Task Analysis:** Observing end users in their operational environment
  - **Scenarios:** Describe the use of the system as a series of interactions between a concrete end user and the system
  - **Use cases:** Abstractions that describe a class of scenarios.

# Scenario-Based Design

Scenarios can have many different uses during the software lifecycle:

- ***Requirements Elicitation***: As-is scenario, visionary scenario
- ***Client Acceptance Test:*** Evaluation scenario
- ***System Deployment:*** Training scenario

Scenario-Based Design: The use of scenarios in a software lifecycle activity

# Types of Scenarios

- As-is scenario:
  - Describes a current situation. Usually used in re-engineering projects. The user describes the system
- Visionary scenario:
  - Describes a future system. Usually used in Greenfield engineering and reengineering projects
  - Can often not be done by the user or developer alone

# Additional Types of Scenarios (2)

- Evaluation scenario:
  - Description of a user task against which the system is to be evaluated.
- Training scenario:
  - A description of the step by step instructions that guide a novice user through a system

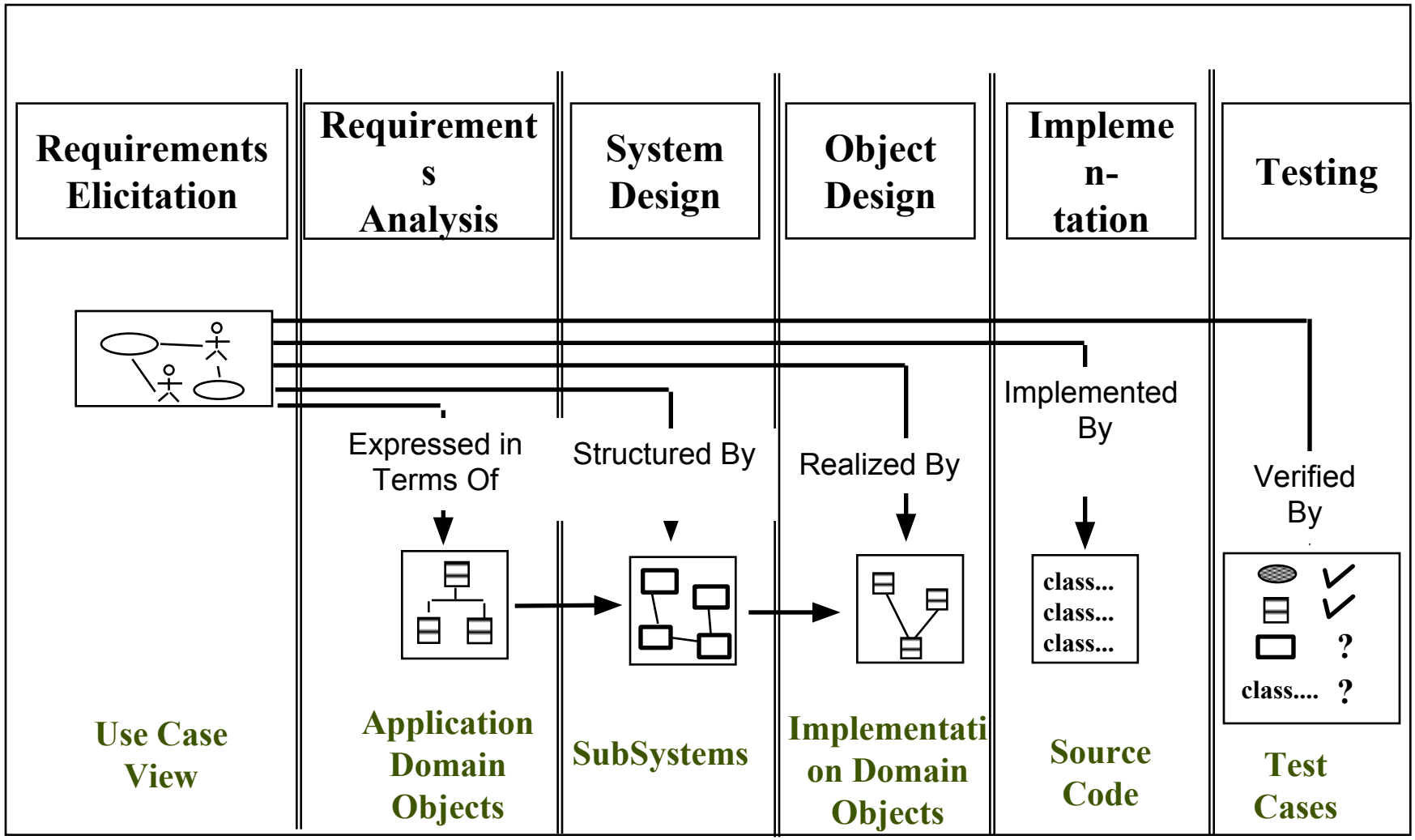
# How do we find scenarios?

- Don’t expect the client to be verbal if the system does not exist
  - Client understands problem domain, not the solution domain.
- Don’t wait for information even if the system exists
  - “What is obvious does not need to be said”
- Engage in a dialectic approach
  - You help the client to formulate the requirements
  - The client helps you to understand the requirements
  - The requirements evolve while the scenarios are being developed

# Heuristics for finding scenarios

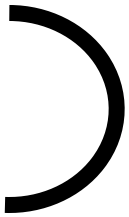
- Ask yourself or the client the following questions:
  - What are the primary tasks that the system needs to perform?
  - What data will the actor create, store, change, remove or add in the system?
  - What external changes does the system need to know about?
  - What changes or events will the actor of the system need to be informed about?
- However, don't rely on questions *and* questionnaires alone
- Insist on task observation if the system already exists (interface engineering or reengineering)
  - Ask to speak to the end user, not just to the client
  - Expect resistance and try to overcome it.

# Software Lifecycle Activities



# Requirements Analysis Document Template

1. Introduction
2. Current system
3. Proposed system
   - 3.1 Overview
   - 3.2 Functional requirements
   - 3.3 Nonfunctional requirements
   - 3.4 Constraints (“Pseudo requirements”)
   - 3.5 System models
     - 3.5.1 Scenarios
     - 3.5.2 Use case model
     - 3.5.3 Object model
       - 3.5.3.1 Data dictionary
       - 3.5.3.2 Class diagrams
     - 3.5.4 Dynamic models
     - 3.5.5 User interface
4. Glossary



# Requirement Document Desiderata

- Correct
  - The client agrees that it represents the reality
- Complete
  - The client agrees that all relevant scenarios are described
- Consistent
  - No two requirements of the specification contradict each other.
- Unambiguous
  - There is only 1 way to interpret the specification
- Realistic
  - All features can be implemented subject to all constraints
- Verifiable
  - Requirements & constraints are testable
- Traceable
  - For each system function there is some requirement[s]
  - For each requirement there is some system function[s]

# Identify Scenarios

Scenario

- A description of what actors do as they use the system
- For each scenario, there is a:
  - Use case
  - Acceptance test case

Questions for identifying scenarios:

- What tasks want actors to perform the system?
- What data does the actor need?
  - Who creates, modifies, removes that data?
- Which external changes affect the system's state?
  - How is the change communicated to the system? (what actor?)
  - Under what circumstances?

# Identify Scenarios - Warehouse on Fire

**Scenario name** warehouseOnFire

**Participating actor instances** bob, alice:FieldOfficer
john:Dispatcher

**Flow of events**

1. Bob, driving down main street in his patrol car, notices smoke coming out
of a warehouse. His partner, Alice, activates the "Report Emergency" function from her FRIEND laptop.

2. Alice enters the address of the building, a brief description of its location (i.e., northwest corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene, given that the area appears to be relatively busy. She confirms her input and waits for an acknowledgment.

# Identify Scenarios - cont

**Flow of events**

3. John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice.

4. Alice receives the acknowledgment and the ETA.

# Observations about Warehouse on Fire Scenario

- Concrete scenario
  - Describes a single instance of reporting a fire incident.
  - Does not describe all possible situations in which a fire can be reported.
- Participating actors
  - Bob, Alice and  John

# Identify Actors

Questions whose answers identify the actors:

- Which user groups does the system support to do their work?
- Which user groups execute the system's primary functions?
- Which user groups execute the system's secondary functions?
  - E.g., maintain or administer the system
- With which external systems does the system interact?
  - E.g., hardware or software

# Identifying Use Cases

| | |
|---|---|
| **Use case name** | ReportEmergency |
| **Participating Actors** | Initiated by FieldOfficer<br>Communicates with Dispatcher |
| **Flow of events** | 1. The FieldOfficer activates the "Report Emergency" function of her terminal.<br>2. FRIEND responds by presenting a form to the FieldOfficer.<br>3. The FieldOfficer completes the form by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. Once the form is completed, the FieldOfficer submits the form.<br>4. FRIEND receives the form and notifies the Dispatcher. |

# Identifying Use Cases_2

**Flow of events**

5. The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. The Dispatcher selects a response and acknowledges the report.

    6. FRIEND displays the acknowledgment and the selected response to the FieldOfficer.

**Entry condition**

The FieldOfficer is logged into FRIEND.

**Exit conditions**

- The FieldOfficer has received an acknowledgment and the selected response from the Dispatcher, OR
- The FieldOfficer has received an explanation indicating why the transaction could not be processed

# Identifying Use Cases_3

**Quality requirements**

- The FieldOfficer's report is acknowledged within 30 seconds.
- The selected response arrives no later than 30 seconds after it is sent by the Dispatcher.

# Simple Use Case Writing Guide

• Use cases should be named with verb phrases. The name of the use case should indicate what the user is trying to accomplish (e.g., ReportEmergency, OpenIncident).

• Actors should be named with noun phrases (e.g., FieldOfficer, Dispatcher, Victim).

• The boundary of the system should be clear. Steps accomplished by the actor and steps accomplished
by the system should be distinguished (e.g., system actions are indented to the right).

• Use case steps in the flow of events should be phrased in the active voice. This makes it explicit who accomplished the step.

• The causal relationship between successive steps should be clear.

# Simple Use Case Writing Guide_cont

• A use case should describe a complete user transaction (e.g., the ReportEmergency use case describes all the steps between initiating the emergency reporting and receiving an acknowledgment).

• Exceptions should be described separately.

• A use case should not describe the user interface of the system. This takes away the focus from the actual steps accomplished by the user and is better addressed with visual mock-ups (e.g., the ReportEmergency only refers to the "Report Emergency" function, not the menu, the button, nor the actual command that corresponds to this function).

# Simple Use Case Writing Guide_cont

- A use case should not exceed two or three pages in length. Otherwise, use include and extend relationships to decompose it in smaller use cases.

# Refine the Use case view

Iterate the following steps:

- Define a horizontal slice (i.e., many high-level scenarios) to define the scope of the system.
- Validate the use case view with the user.
- Detail a vertical slice.
- Validate with the user.
- In extreme  programming, you design, implement, test [and integrate] this vertical slice, before tackling the next vertical slice.

# Identify Relationships Among Actors & Use cases

- Access control – which actors access which functionality – is represented with use cases.
- Extend relation
  - ConnectionDown use case:
    - Entry condition:
      - Connection between FieldOfficer & Dispatcher fails before EmergencyReport is submitted
    - Event flow:
      - FieldOfficer notifies Dispatcher via voice channel
- Heuristics:
  - Use extend relation for exceptions, optional, or rare behavior.
  - Use include relation for behavior common to 2 or more use cases.

# Identify Initial Analysis Objects

It may be one of the system objects if it is a:

- Term developers/users need to clarify to understand a use case;
- Recurring noun in the use case model;
- Real-world object the system needs to track (e.g., FieldOfficer)
- Real-world process the system needs to track (e.g., EmergencyOperationPlan)
- Use cases (e.g., ReportEmergency)
- Application domain term

# Identify Nonfunctional Requirements

Investigate the following:

- User interface – level of expertise of user
- Documentation – User manual? The system design should be documented. What about development process?
- Hardware considerations – What assumptions are made?
- Error handling – philosophy (e.g., tolerates failure of any single system component).
- Physical environment – what assumptions are made about power supply, cooling, etc.
- Physical security
- Resource – constraints on power, memory, etc.

# Example of questions for eliciting nonfunctional requirements

| | |
|---|---|
| **Supportability** *(including maintainability and portability)* | • What are the foreseen extensions to the system?<br>• Who maintains the system?<br>• Are there plans to port the system to different software or hardware environments? |
| **Implementation** | • Are there constraints on the hardware platform?<br>• Are constraints imposed by the maintenance team?<br>• Are constraints imposed by the testing team? |
| **Interface** | • Should the system interact with any existing systems?<br>• How are data exported/imported into the system?<br>• What standards in use by the client should be supported by the system? |

# Example of questions for eliciting nonfunctional requirements

| | |
|---|---|
| **Operation** | • Who manages the running system? |
| **Packaging** | • Who installs the system?<br>• How many installations are foreseen?<br>• Are there time constraints on the installation? |
| **Legal** | • How should the system be licensed?<br>• Are any liability issues associated with system failures?<br>• Are any royalties or licensing fees incurred by using specific algorithms or components? |

In practice, analysts use a taxonomy of nonfunctional requirements (e.g., the FURPS+) to generate check lists of questions to help the client and the developers focus on the nonfunctional aspects of the system. **Functionality, Usability, Reliability, Performance, and Supportability**

# To Do List

- Project definition
  - Complete a project description that the customer agrees to.
- Research
  - Initial identification of actors & event flows
  - Document unresolved issues/ambiguities
- Create a draft Requirements Analysis Document (RAD)
- While (there are unresolved items) do:
  - Resolve an item on list of issues/ambiguities;
  - Modify RAD accordingly;
  - Insert new, more detailed unresolved items;

# Requirements Elicitation: Difficulties and Challenges

- Communicate accurately about the domain and the system
  - People with different backgrounds must collaborate to bridge the gap between end users and developers
    - Client and end users have application domain knowledge
    - Developers have solution domain knowledge
- Identify an appropriate system (Defining the system boundary)
- Provide an unambiguous specification
- Leave out unintended features

# Example of an Ambiguous Specification

During a laser experiment, a laser beam was directed from earth to a mirror on the Space Shuttle Discovery

The laser beam was supposed to be reflected back towards a mountain top 10,023 feet high

The operator entered the elevation as “10023”

The light beam never hit the mountain top
What was the problem?

# Example of an Ambiguous Specification

During a laser experiment, a laser beam was directed from earth to a mirror on the Space Shuttle Discovery

The laser beam was supposed to be reflected back towards a mountain top 10,023 feet high

The operator entered the elevation as “10023”

The light beam never hit the mountain top
What was the problem?

The computer interpreted the number in miles...

# Example of an Unintended Feature

**From the News: London underground train leaves station without driver!**

What happened?

- A passenger door was stuck and did not close
- The driver left his train to close the passenger door
  - He left the driver door open
  - He relied on the specification that said the train does not move if at least one door is open
- When he shut the passenger door, the train left the station without him

# Example of an Unintended Feature

**From the News: London underground train leaves station without driver!**

What happened?

- A passenger door was stuck and did not close
- The driver left his train to close the passenger door
  - He left the driver door open
  - He relied on the specification that said the train does not move if at least one door is open
- When he shut the passenger door, the train left the station without him
- The driver door was not treated as a door in the source code!

# Scenario example from earlier: Warehouse on Fire

- Bob, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, Alice, reports the emergency from her car.

- Alice enters the address of the building into her wearable computer , a brief description of its location (i.e., north west corner), and an emergency level.
- 
- She confirms her input and waits for an acknowledgment.
- John, the dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and sends the estimated arrival time (ETA) to Alice.

- Alice received the acknowledgment and the ETA.

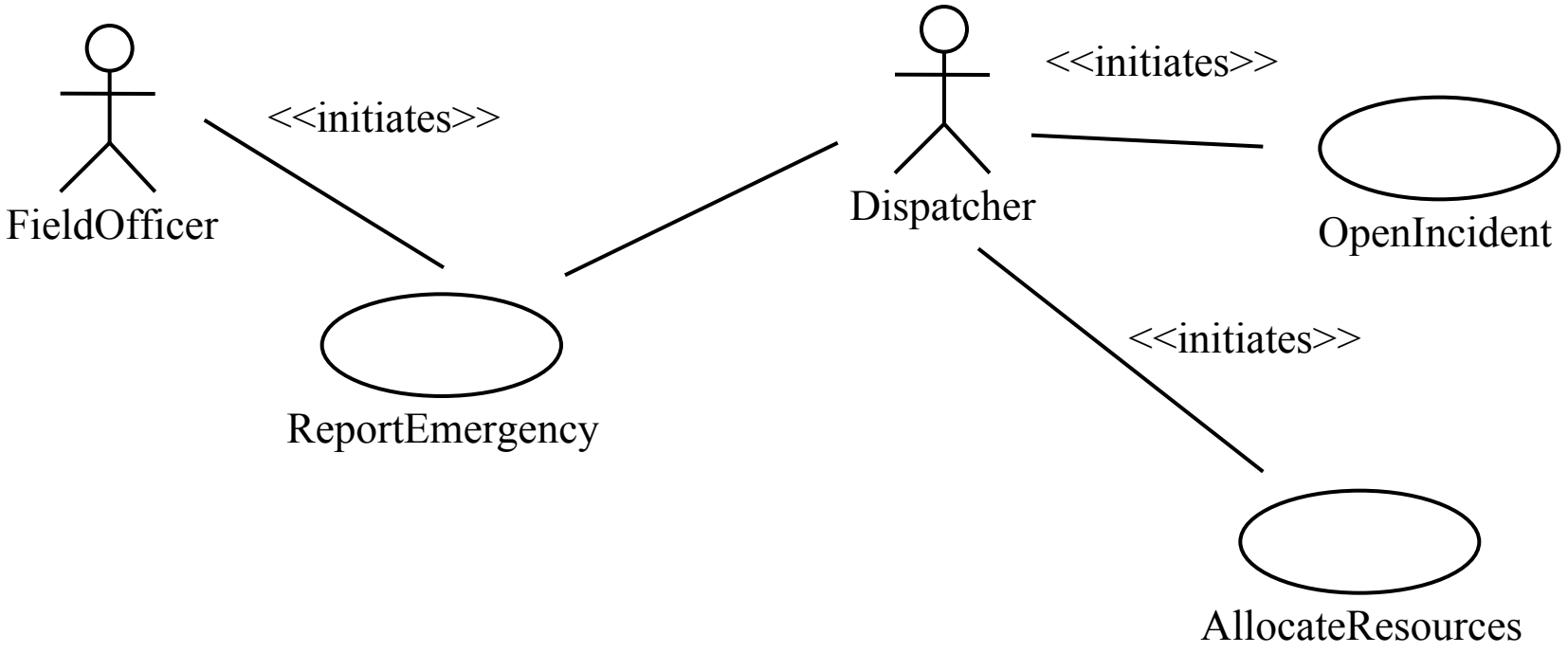
# Observations about Warehouse on Fire Scenario

- Concrete scenario
  - Describes a single instance of reporting a fire incident.
  - Does not describe all possible situations in which a fire can be reported.
- Participating actors
  - Bob, Alice and  John

# After the scenarios are formulated

- Find all the use cases in the scenario that specify all instances of how to report a fire
  - Example: “Report Emergency“ in the first paragraph of the scenario is a candidate for a use case
- Describe each of these use cases in more detail
  - Participating actors
  - Describe the entry condition
  - Describe the flow of events
  - Describe the exit condition
  - Describe exceptions
  - Describe nonfunctional requirements

# Use Case View for Incident Management



# How to find Use Cases

- Select a narrow vertical slice of the system (i.e. one scenario)
  - Discuss it in detail with the user to understand the user's preferred style of interaction
- Select a horizontal slice (i.e. many scenarios) to define the scope of the system.
  - Discuss the scope with the user
- Use illustrative prototypes (mock-ups) as visual support
- Find out what the user does
  - Task observation (Good)
  - Questionnaires (Bad)

# Use Case Example: ReportEmergency

- Use case name: ReportEmergency
- Participating Actors:
  - Field Officer (Bob and Alice in the Scenario)
  - Dispatcher (John in the Scenario)
- Exceptions:
  - The FieldOfficer is notified immediately if the connection between terminal and central is lost.
  - The Dispatcher is notified immediately if the connection between a FieldOfficer and central is lost.
- Flow of Events: **on next slide.**
- Special Requirements:
  - The Field Officer's report is acknowledged within 30 seconds. The selected response arrives no later than 30 seconds after it is sent by the Dispatcher.

# Use Case Example: ReportEmergency

## Flow of Events

1. The **FieldOfficer** activates the "Report Emergency" function of her terminal. FRIEND responds by presenting a form to the officer.
2. The FieldOfficer fills the form, by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes a response to the emergency situation. Once the form is completed, the FieldOfficer submits the form, and the **Dispatcher** is notified.
3. The Dispatcher creates an Incident in the database by invoking the OpenIncident use case. He selects a response and acknowledges the report.
4. The FieldOfficer receives the acknowledgment and the selected response.

# Order of steps when formulating use cases

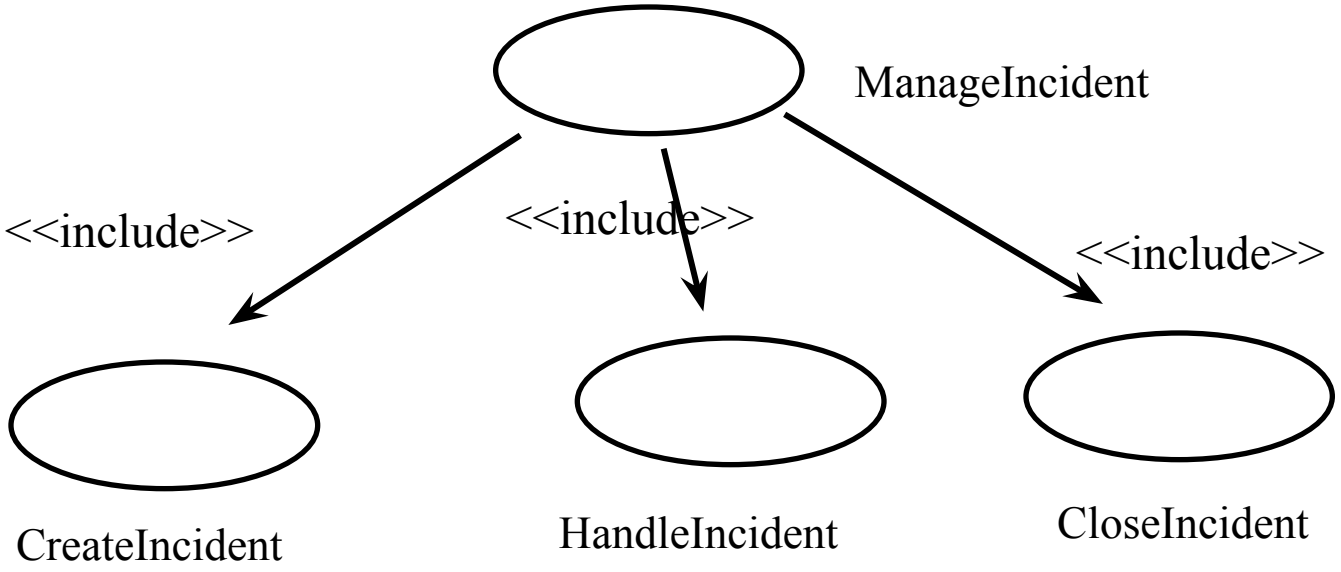
- First step: Name the use case
  - Use case name: ReportEmergency
- Second step: Find the actors
  - Generalize the concrete names (“Bob”) to participating actors (“Field officer”)
  - Participating Actors:
    - Field Officer (Bob and Alice in the Scenario)
    - Dispatcher (John in the Scenario)
- Third step: Concentrate on the flow of events
  - Use informal natural language

# Use Case Associations

- Dependencies between use cases are represented with use case associations
- Associations are used to reduce complexity
  - Decompose a long use case into shorter ones
  - Separate alternate flows of events
  - Refine abstract use cases
- Types of use case associations
  - Includes
  - Extends
  - Generalization

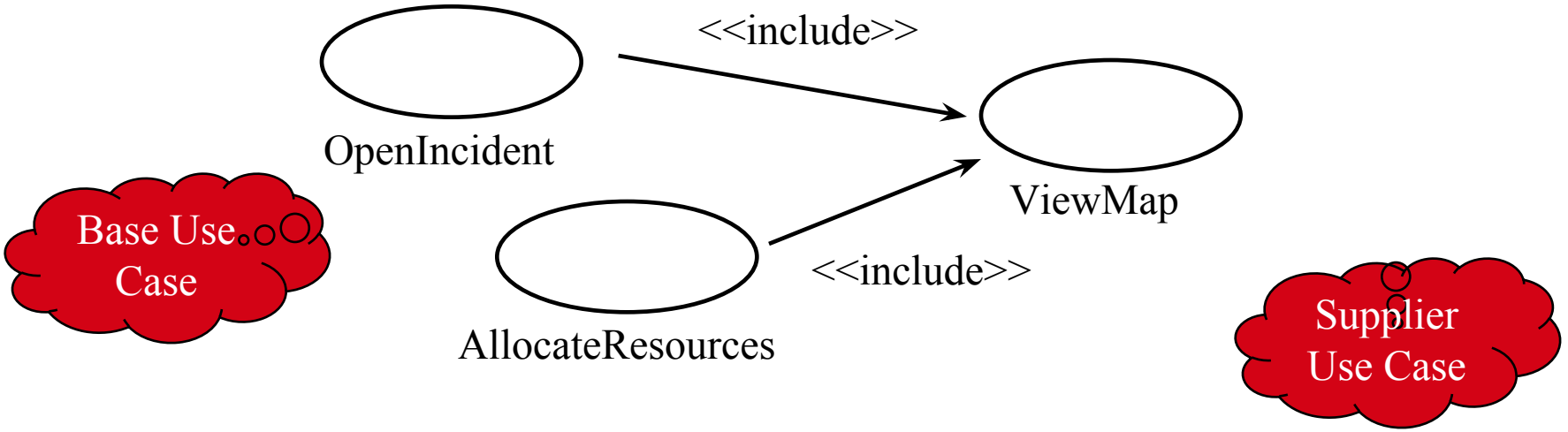
# <<include>>: Functional Decomposition

- Problem:
  - A function in the original problem statement is too complex
- Solution:
  - Describe the function as the aggregation of a set of simpler functions. The associated use case is decomposed into shorter use cases



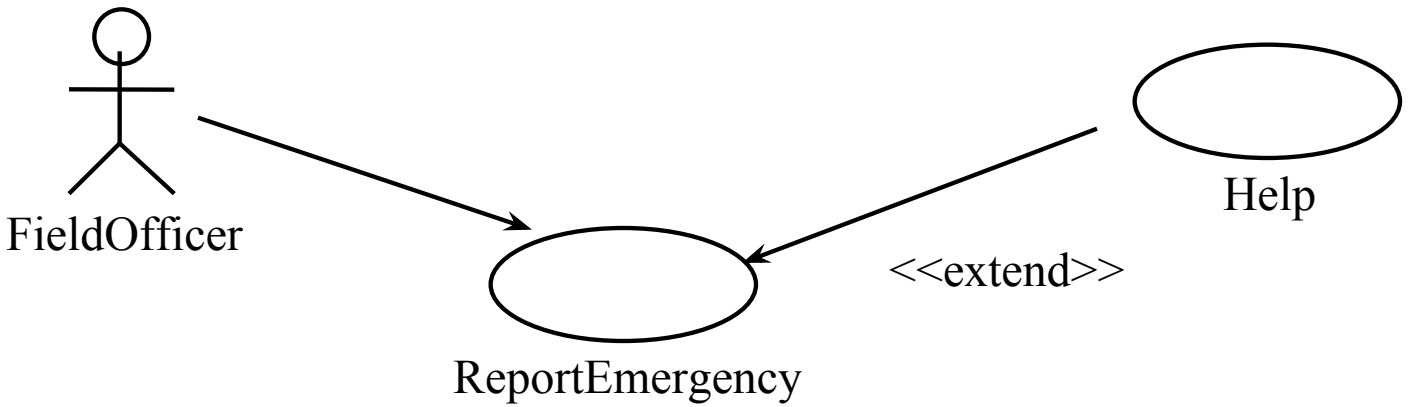
# <<include>>: Reuse of Existing Functionality

- Problem: There are overlaps among use cases. How can we *reuse* flows of events instead of duplicating them?
- Solution: The *includes association* from use case A to use case B indicates that an instance of use case A performs all the behavior described in use case B ("A delegates to B")
- Example: Use case "ViewMap" describes behavior that can be used by use case "OpenIncident" ("ViewMap" is factored out)



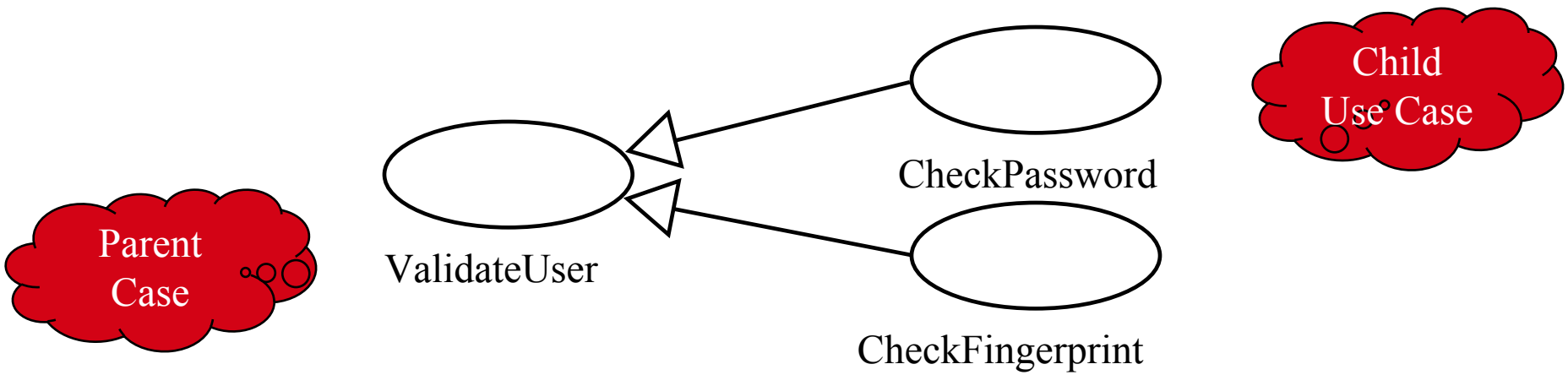
# <<extend>> Association for Use Cases

- Problem: The functionality in the original problem statement needs to be extended.
- Solution: An *extend association* from use case A to use case B
- Example: "ReportEmergency" is complete by itself, but may be extended by use case "Help" for a scenario in which the user requires help



# Generalization in Use Cases

- Problem: We want to factor out common (but not identical) behavior.
- Solution: The child use cases inherit the behavior and meaning of the parent use case and add or override some behavior.
- Example: "ValidateUser" is responsible for verifying the identity of the user. The customer might require two realizations: "CheckPassword" and "CheckFingerprint"



# Another Use Case Example

**Actor Bank Customer**

- Person who owns one or more Accounts in the Bank.

**Withdraw Money**

- The Bank Customer specifies a Account and provides credentials to the Bank proving that s/he is authorized to access the Bank Account.
- The Bank Customer specifies the amount of money s/he wishes to withdraw.
- The Bank checks if the amount is consistent with the rules of the Bank and the state of the Bank Customer's account. If that is the case, the Bank Customer receives the money in cash.

# Use Case Attributes

Use Case **Withdraw Money Using ATM**

Initiatiating actor:

- Bank Customer

Preconditions:

- Bank Customer has opened a Bank Account with the Bank ***and***
- Bank Customer has received an ATM Card and PIN

Postconditions:

- Bank Customer has the requested cash ***or***
- Bank Customer receives an explanation from the ATM about why the cash could not be dispensed

# Use Case Flow of Events

| Actor steps | System steps |
| --- | --- |
| 1.The Bank Customer inputs the card into the ATM. | |
| | 2.The ATM requests the input of a four-digit PIN. |
| 3. The Bank Customer types in PIN. | |
| | 4. If several accounts are recorded on the card, the ATM offers a choice of the account numbers for selection by the Bank Customer |
| 5. The Bank Customer selects an account. | |
| | 6.If only one account is recorded on the card or after the selection, the ATM requests the amount to be withdrawn. |
| 7. The Bank Customer inputs an amount. | |
| | 8.The ATM outputs the money and a receipt and stops the interaction. |

# Use Case Exceptions

**Actor steps**

1. The Bank Customer inputs her card into the ATM.**[Invalid card]**

3. The Bank Customer types in PIN. **[Invalid PIN]**

5. The Bank Customer selects an account .

7. The Bank Customer inputs an amount. **[Amount over limit]**

[Invalid card]
The ATM outputs the card and stops the interaction.

[Invalid PIN]
The ATM announces the failure and offers a 2nd try as well as canceling the whole use case. After 3 failures, it announces the possible retention of the card. After the 4th failure it keeps the card and stops the interaction.

[Amount over limit]
The ATM announces the failure and the available limit and offers a second try as well as canceling the whole use case.

# Guidelines for Formulation of Use Cases (1)

- Name
  - Use a verb phrase to name the use case.
  - The name should indicate what the user is trying to accomplish.
  - Examples:
    - "Request Meeting", "Schedule Meeting", "Propose Alternate Date"
- Length
  - A use case description should not exceed 1-2 pages. If longer, use include relationships.
  - A use case should describe a complete set of interactions.

# Guidelines for Formulation of Use Cases (2)

Flow of events:

- Use the active voice. Steps should start either with “The Actor” or “The System …”.
- The causal relationship between the steps should be clear.
- All flow of events should be described (not only the main flow of event).
- The boundaries of the system should be clear. Components external to the system should be described as such.
- Define important terms in the glossary.

# Example of a badly written Use Case

"The driver arrives at the parking gate, the driver receives a ticket from the distributor, the gate is opened, the driver drives through."

- What is wrong with this use case?

# Example of a badly written Use Case

"The driver arrives at the parking gate, the driver receives a ticket from the distributor, the gate is opened, the driver drives through."

It contains no actors

It is not clear which action triggers the ticket being issued

Because of the passive form, it is not clear who opens the gate (The driver? The computer? A gate keeper?)

It is not a complete transaction. A complete transaction would also describe the driver paying for the parking and driving out of the parking lot.

# How to write a use case (Summary)

- Name of Use Case
- Actors
  - Description of Actors involved in use case
- Entry condition
  - "This use case starts when…"
- Flow of Events
  - Free form,  informal natural language
- Exit condition
  - "This use cases terminates when…"
- Exceptions
  - Describe what happens if things go wrong
- Special Requirements
  - Nonfunctional Requirements, Constraints

# Summary

- Scenarios:
  - Great way to establish communication with client
  - Different types of scenarios: As-Is, visionary, evaluation and training
- Use cases
  - Abstractions of scenarios
- Use cases bridge the transition between functional requirements and objects.