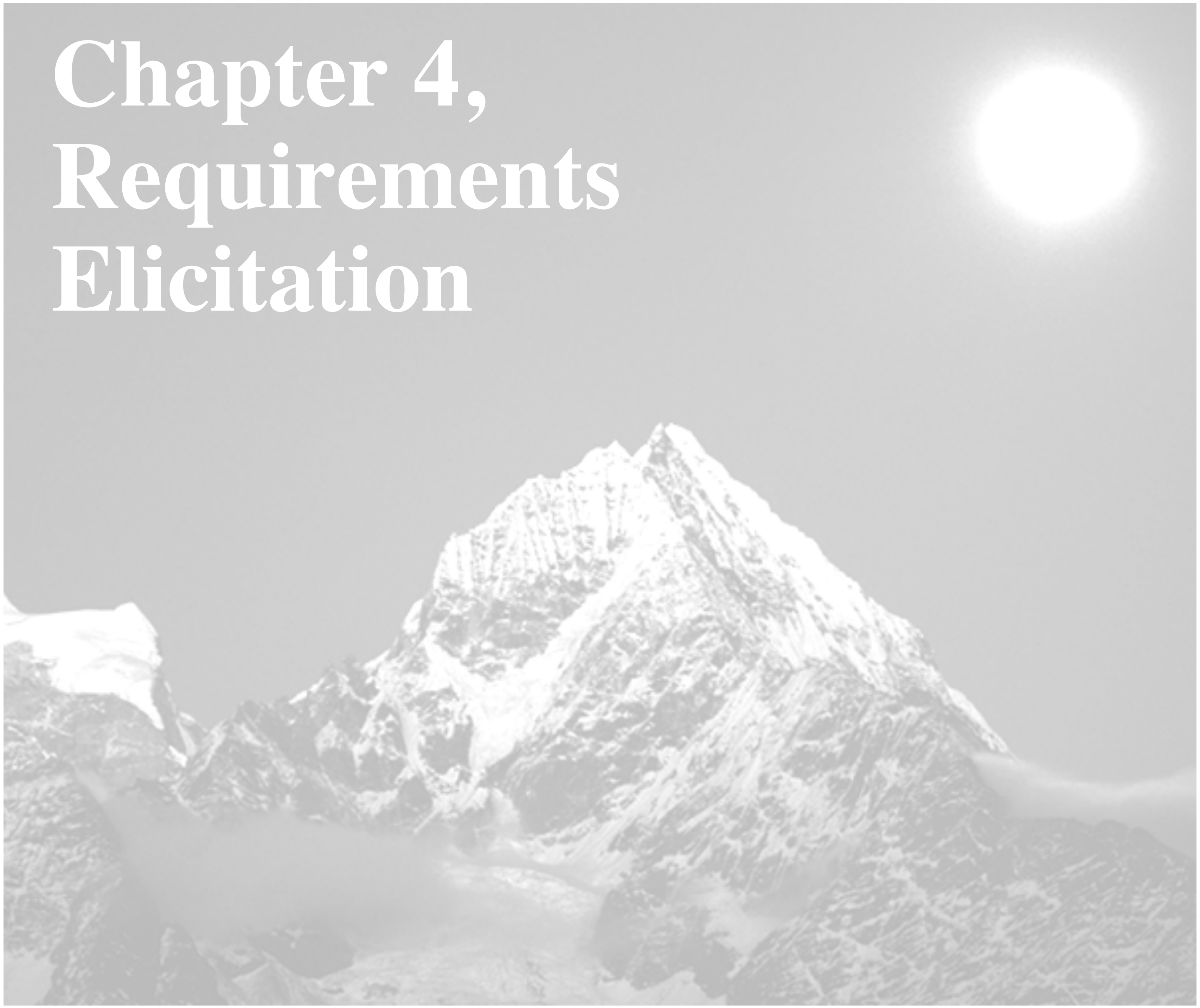
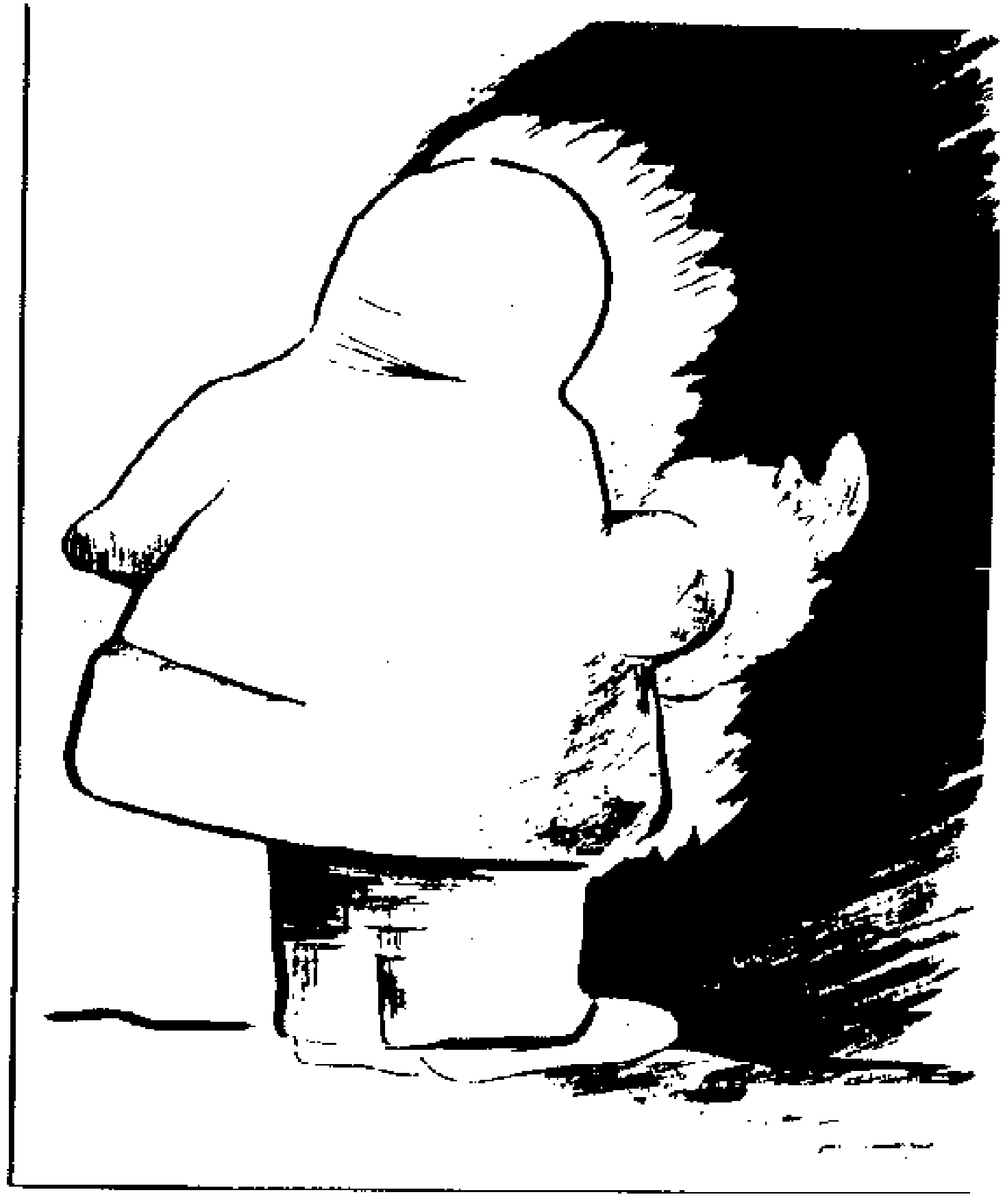
Object-Oriented Software Engineering
Using UML, Patterns, and Java

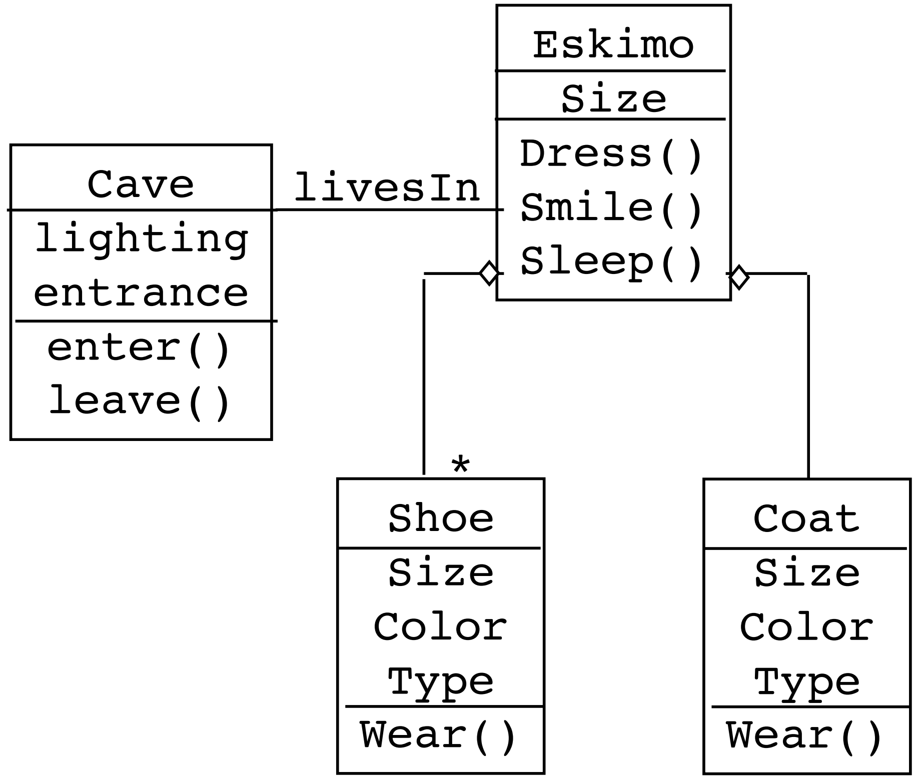
Chapter 4, Requirements Elicitation



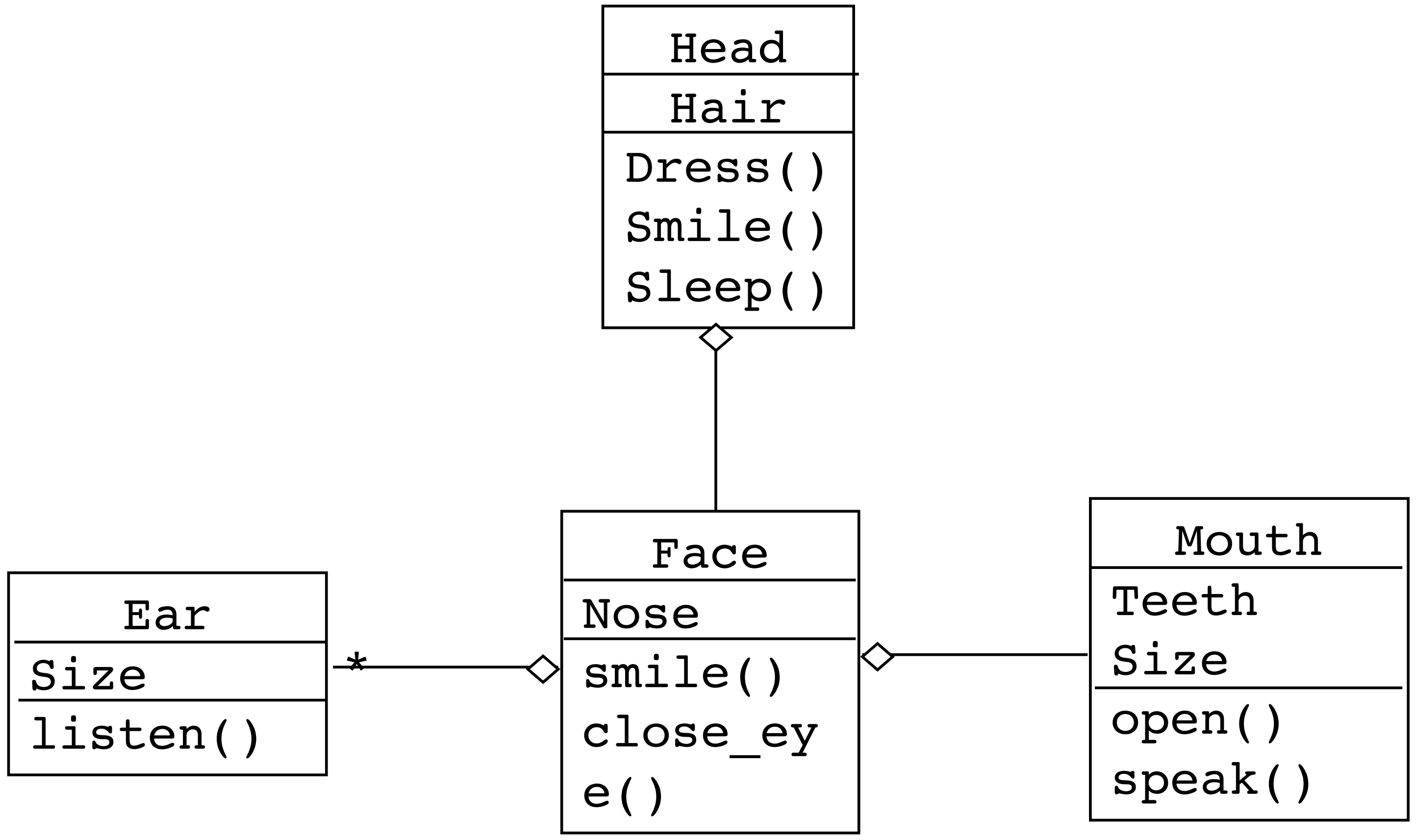
What is This?



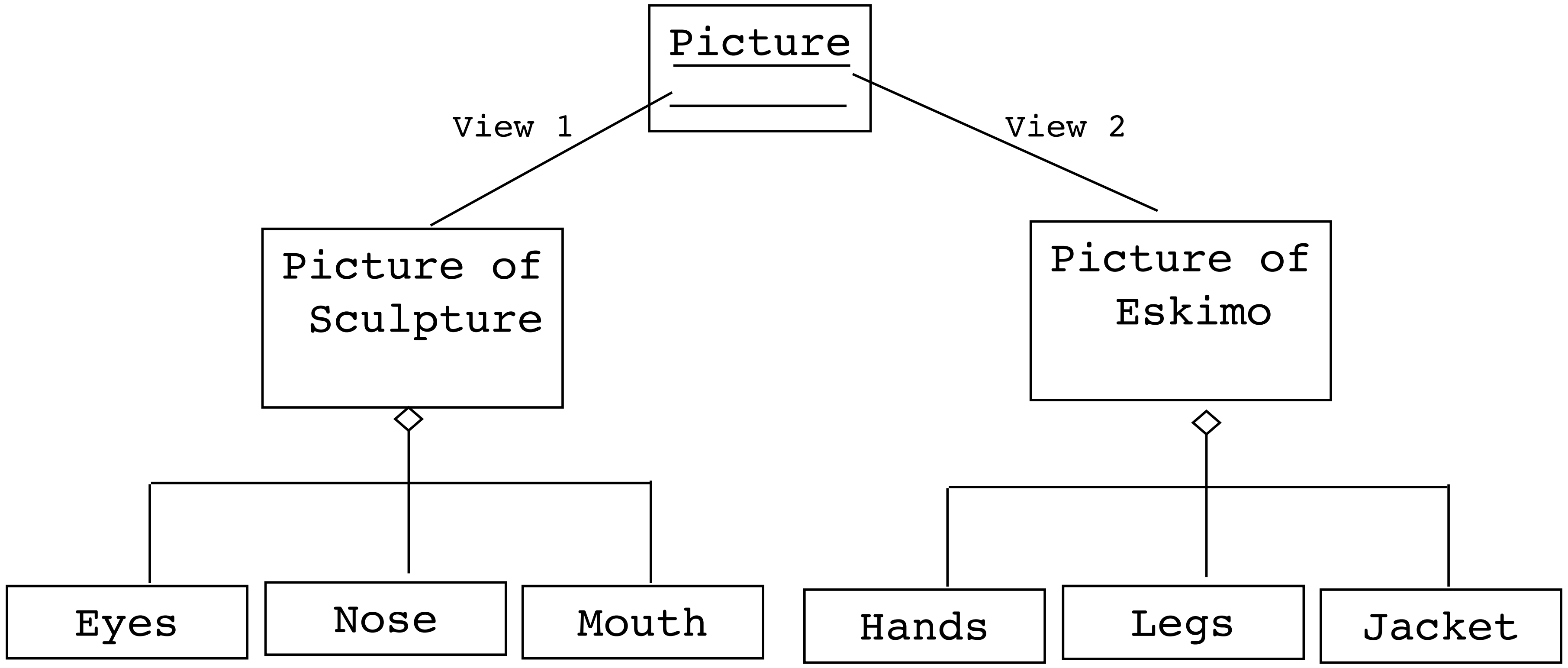
Possible Object Model: Eskimo



Alternative: Head



The Artist's View



Where are we right now?

- Three ways to deal with complexity:
  - **Abstraction**
  - **Decomposition (Technique: Divide and conquer)**
  - **Hierarchy (Technique: Layering)**
- Two ways to deal with decomposition:
  - **Object-orientation and functional decomposition**
  - **Functional decomposition leads to unmaintainable code**
  - **Depending on the purpose of the system, different objects can be found**
- What is the right way?
  - **Start with a description of the functionality (Use case model). Then proceed by finding objects (object model).**
- What activities and models are needed?
  - **This leads us to the software lifecycle we use in this class**

Software Lifecycle Definition

- Software lifecycle:
  - **Set of** activities **and their relationships to each other to support the development of a software system**
- Typical Lifecycle questions:
  - **Which activities should I select for the software project?**
  - **What are the dependencies between activities?**
  - **How should I schedule the activities?**
  - **What is the result of an activity?**

Example: Selection of Software Lifecycle Activities for a specific project

The Hacker knows only one activitity

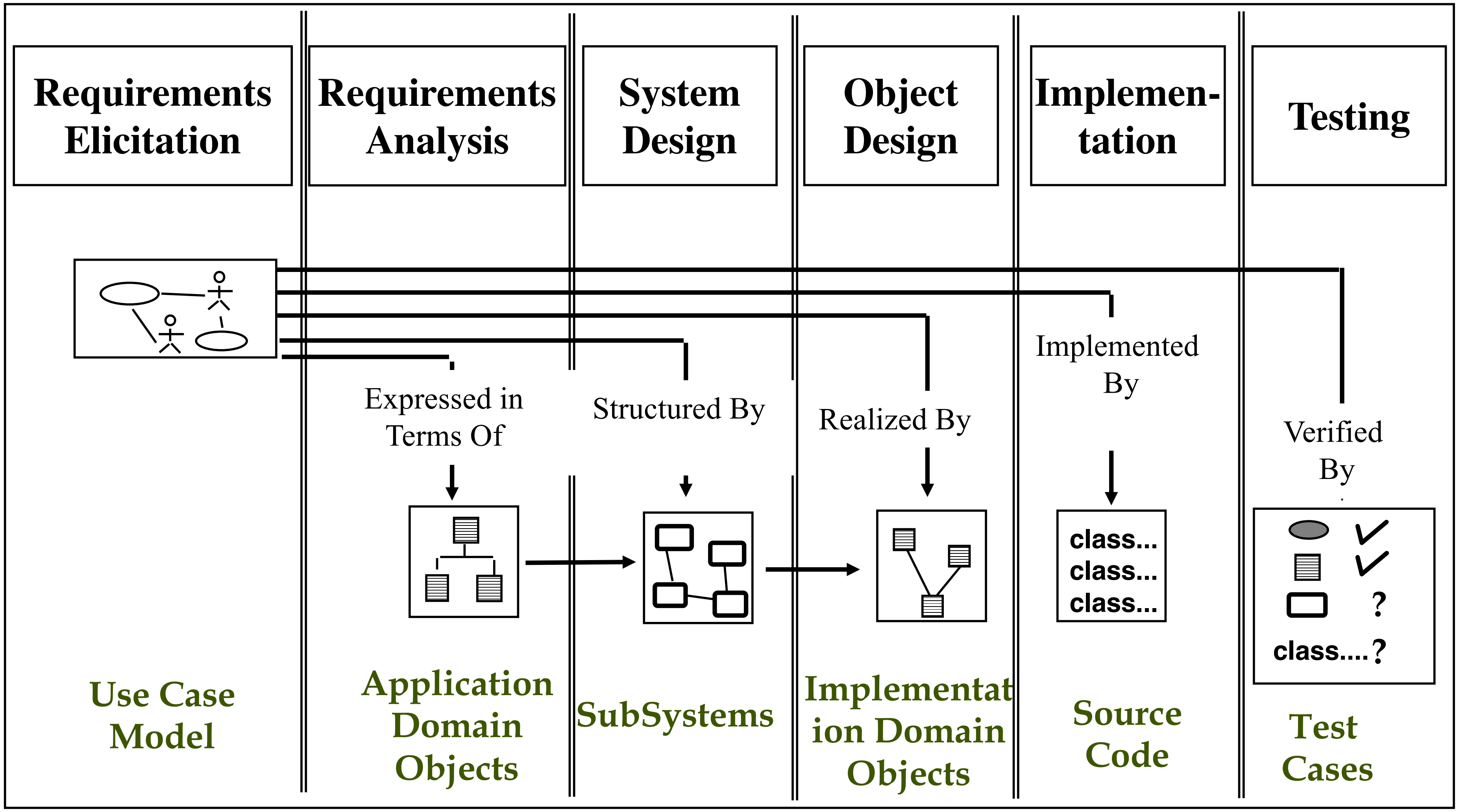
Implemen-tation

Activities used this lecture

Requirements Elicitation | **Analysis** | **System Design** | **Object Design** | **Implemen-tation** | **Testing**

Each activity produces one or more models

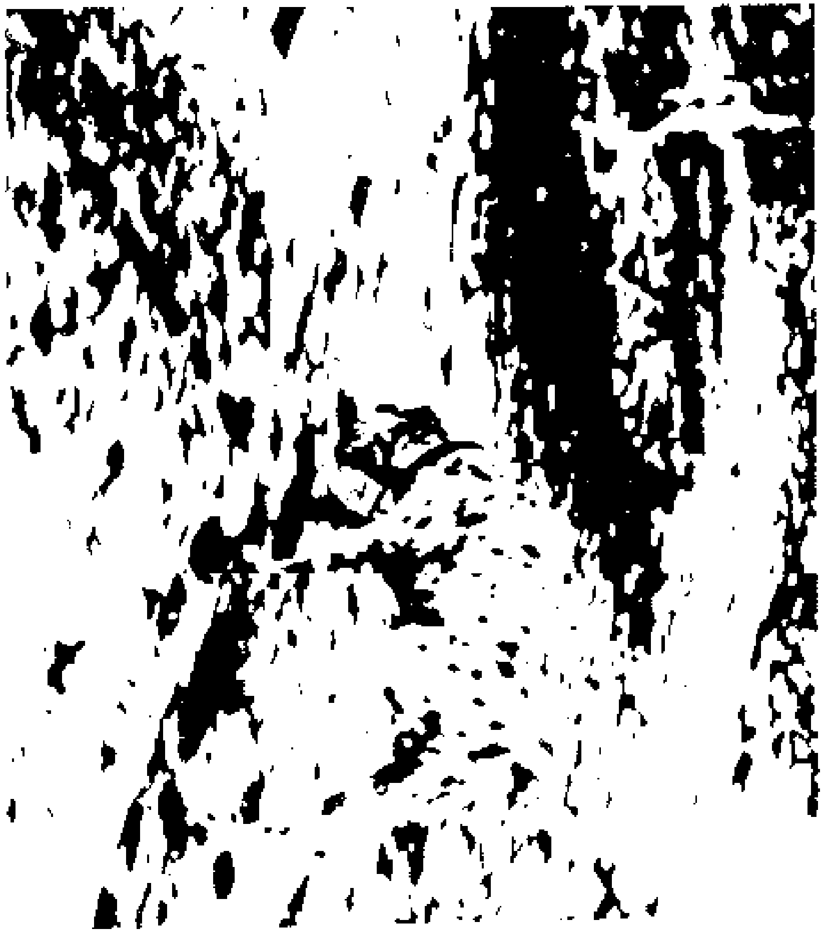
Software Lifecycle Activities



First Step in Establishing the Requirements: System Identification

- The development of a system is not just done by taking a snapshot of a scene (domain)
- Two questions need to be answered:
  - **How can we identify the purpose of a system?**
  - **Crucial is the definition of the system boundary: What is inside, what is outside the system?**
- These two questions are answered in the requirements process
- The requirements process consists of two activities:
  - **Requirements Elicitation:**
    - **Definition of the system in terms understood by the customer ("Problem Description")**
  - **Requirements Analysis:**
    - **Technical specification of the system in terms understood by the developer ("Problem Specification")**

Defining the System Boundary: What do you see?



Example of an Ambiguous Specification

During an 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...

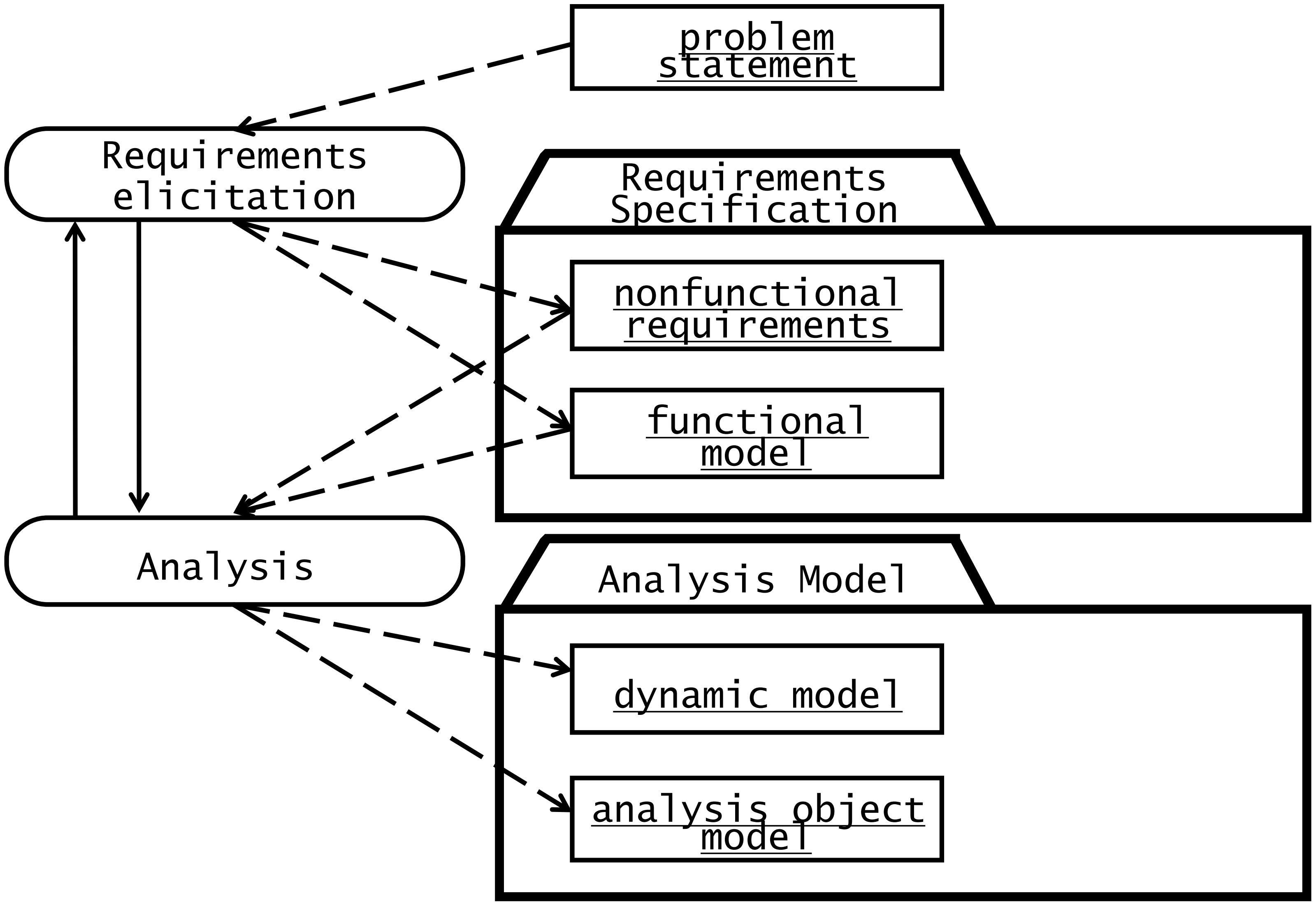
Example of an Unintended Feature

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.
    - The driver door was not treated as a door in the source code!

Products of Requirements Process



System Specification vs Analysis Model

- Both models focus on the requirements from the user's view of the system.
- ***System specification*** uses natural language (derived from the *problem statement*)
- The ***analysis model*** uses formal or semi-formal notation (for example, a graphical language like UML)

- The starting point is the problem statement

Problem Statement

- The problem statement is developed by the client as a description of the problem addressed by the system
- Other words for problem statement:
    - **Statement of Work**
- A good problem statement describes
    - **The current situation**
    - **The functionality the new system should support**
    - **The environment in which the system will be deployed**
    - **Deliverables expected by the client**
    - **Delivery dates**
    - **A set of acceptance criteria**

Ingredients of a Problem Statement

- Current situation: The Problem to be solved
- Description of one or more scenarios
- Requirements
  - **Functional and Nonfunctional requirements**
  - **Constraints ("pseudo requirements")**
- Project Schedule
  - **Major milestones that involve interaction with the client including deadline for delivery of the system**
- Target environment
  - **The environment in which the delivered system has to perform a specified set of system tests**
- Client Acceptance Criteria
  - **Criteria for the system tests**

Current Situation: The Problem To Be Solved

- There is a problem in the current situation
  - **Examples:**
    - **The response time when playing letter-chess is far too slow.**
    - **I want to play Go, but cannot find players on my level.**
- What has changed? Why can address the problem now?
  - **There has been a change, either in the application domain or in the solution domain**
  - *Change in the application domain*
    - **A new function (business process) is introduced into the business**
    - **Example: We can play highly interactive games with remote people**
  - *Change in the solution domain*
    - **A new solution (technology enabler) has appeared**
    - **Example: The internet allows the creation of virtual communities.**

ARENA: The Problem

- The Internet has enabled virtual communities
    - **Groups of people sharing common of interests but who have never met each other in person. Such virtual communities can be short lived (e.g people in a chat room or playing a multi player game) or long lived (e.g., subscribers to a mailing list).**
- Many multi-player computer games now include support for virtual communities.
    - **Players can receive news about game upgrades, new game levels, announce and organize matches, and compare scores.**
- Currently each game company develops such community support in each individual game.
    - **Each company uses a different infrastructure, different concepts, and provides different levels of support.**
- This redundancy and inconsistency leads to problems:
    - **High learning curve for players joining a new community,**
    - **Game companies need to develop the support from scratch**
    - **Advertisers need to contact each individual community separately.**

ARENA: The Objectives

- Provide a generic infrastructure for operating an arena to
  - **Support virtual game communities.**
  - **Register new games**
  - **Register new players**
  - **Organize tournaments**
  - **Keeping track of the players scores.**
- Provide a framework for tournament organizers
  - **to customize the number and sequence of matchers and the accumulation of expert rating points.**
- Provide a framework for game developers
  - **for developing new games, or for adapting existing games into the ARENA framework.**
- Provide an infrastructure for advertisers.

Types of Requirements

- Functional requirements:
  - **Describe the interactions between the system and its environment independent from implementation**
  - **Examples:**
    - **An ARENA operator should be able to define a new game.**
- Nonfunctional requirements:
  - **User visible aspects of the system not directly related to functional behavior.**
  - **Examples:**
    - **The response time must be less than 1 second**
    - **The ARENA server must be available 24 hours a day**
- Constraints ("Pseudo requirements"):
  - **Imposed by the client or the environment in which the system operates**
    - **The implementation language must be Java**
    - **ARENA must be able to dynamically interface to existing games provided by other game developers.**

What is usually not 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 critical step in the development process, usually after requirements engineering or requirements analysis. Also at delivery (client acceptance test).
- **Requirements validation criteria:**
  - **Correctness:**
    - The requirements represent the client's view.
  - **Completeness:**
    - All possible scenarios, in which the system can be used, are described, including exceptional behavior by the user or the system
  - **Consistency:**
    - There are functional or nonfunctional requirements that contradict each other
  - **Realism:**
    - Requirements can be implemented and delivered
  - **Traceability:**
    - Each system function can be traced to a corresponding set of functional requirements

Requirements Evolution

- Problem with requirements validation: Requirements change very fast during requirements elicitation.
- Tool support for managing requirements:
  - **Store requirements in a shared repository**
  - **Provide multi-user access**
  - **Automatically create a system specification document from the repository**
  - **Allow change management**
  - **Provide traceability throughout the project lifecycle**
- RequisitPro from Rational
  - **http://www.rational.com/products/reqpro/docs/datasheet.html**

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.

Types of Requirements Elicitation

- Greenfield Engineering
  - **Development starts from scratch, no prior system exists, the requirements are extracted from the end users and the client**
  - **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
  - **Provide the services of an existing system in a new environment**
  - **Triggered by technology enabler or new market needs**

Requirements Elicitation

- Very challenging activity
- Requires collaboration of people with different backgrounds
    - **Users with application domain knowledge**
    - **Developer with solution domain knowledge (design knowledge, implementation knowledge)**
- Bridging the gap between user and developer:
    - *Scenarios:* **Example of the use of the system in terms of a series of interactions between the user and the system**
    - *Use cases:* **Abstraction that describes a class of scenarios**

Tecnica	Serve per	Vantaggi	Svantaggi
Questionari	Rispondere a domande specifiche.	Si possono raggiungere molte persone con poco sforzo.	Vanno progettati con grande accuratezza, in caso contrario le risposte potrebbero risultare poco informative. Il tasso di risposta può essere basso.
Interviste individuali	Esplorare determinati aspetti del problema e determinati punti di vista.	L'intervistatore può controllare il corso dell'intervista, orientandola verso quei temi sui quali l'intervistato è in grado di fornire i contributi più utili.	Richiedono molto tempo. Gli intervistati potrebbero evitare di esprimersi con franchezza su alcuni aspetti delicati.
Focus group	Mettere a fuoco un determinato argomento, sul quale possono esserci diversi punti di vista.	Fanno emergere le aree di consenso e di conflitto. Possono far emergere soluzioni condivise dal gruppo.	La loro conduzione richiede esperienza. Possono emergere figure dominanti che monopolizzano la discussione.
Osservazioni sul campo	Comprendere il contesto delle attività dell'utente.	Permettono di ottenere una consapevolezza sull'uso reale del prodotto che le altre tecniche non danno.	Possono essere difficili da effettuare e richiedere molto tempo e risorse.
Suggerimenti spontanei degli utenti	Individuare specifiche necessità di miglioramento di un prodotto.	Hanno bassi costi di raccolta. Possono essere molto specifici.	Hanno normalmente carattere episodico.
Analisi della concorrenza e delle best practices	Individuare le soluzioni migliori adottate nel settore di interesse	Evitare di "reinventare la ruota" e ottenere vantaggio competitivo	L'analisi di solito è costosa (tempo e risorse)

http://www.rpolillo.it/faciledausare/Cap.7.htm

Why Scenarios and Use Cases?

- Utterly comprehensible by the user
  - **Use cases model a system from the users' point of view (functional requirements)**
    - **Define every possible event flow through the system**
    - **Description of interaction between objects**
- Great tools to manage a project. Use cases can form basis for whole development process
  - **User manual**
  - **System design and object design**
  - **Implementation**
  - **Test specification**
  - **Client acceptance test**
- An excellent basis for incremental & iterative development
- Use cases have also been proposed for business process reengineering (Ivar Jacobson)

Scenarios

- "A narrative description of what people do and experience as they try to make use of computer systems and applications" [M. Carrol, Scenario-based Design, Wiley, 1995]
- A concrete, focused, informal description of a single feature of the system used.
- Scenarios can have many different uses during the software lifecycle

Types of Scenarios

- As-is scenario:
  - **Used in describing a current situation. Usually used in re-engineering projects. The user describes the system.**
    - **Example: Description of Letter-Chess**
- Visionary scenario:
  - **Used to describe a future system. Usually used in greenfield engineering and reengineering projects.**
  - **Can often not be done by the user or developer alone**
    - **Example: Description of an interactive internet-based Tic Tac Toe game tournament.**
- Evaluation scenario:
  - **User tasks against which the system is to be evaluated.**
    - **Example: Four users (two novice, two experts) play in a TicTac Toe tournament in ARENA.**
- Training scenario:
  - **Step by step instructions that guide a novice user through a system**
    - **Example: How to play Tic Tac Toe in the ARENA Game Framework.**

How do we find scenarios?

- Don't expect the client to be verbal if the system does not exist (greenfield engineering)
- Don't wait for information even if the system exists
- Engage in a dialectic approach (evolutionary, incremental)
  - **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 *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 software contractor**
  - **Expect resistance and try to overcome it**

Example: Accident Management System

- What needs to be done to report a “Cat in a Tree” incident?
- What do you need to do if a person reports “Warehouse on Fire?”
- Who is involved in reporting an incident?
- What does the system do if no police cars are available? If the police car has an accident on the way to the “cat in a tree” incident?
- What do you need to do if the “Cat in the Tree” turns into a “Grandma has fallen from the Ladder”?
- Can the system cope with a simultaneous incident report “Warehouse on Fire?”

Scenario Example: 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, a brief description of its location (i.e., north west corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene given that area appear to be relatively busy. 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 two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice.
- Alice received the acknowledgment and the ETA.

Observations about Warehouse on Fire Scenario

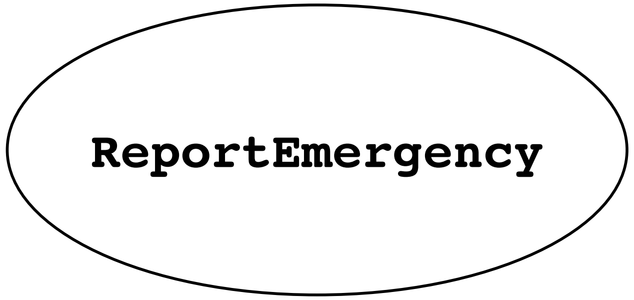
- Concrete scenario
  - **Describes a single instance of reporting a fire incident.**
  - **Does not describe all possible situations in which a fire can be reported.**
- Participating actors
  - **Bob, Alice and John**

Next goal, after the scenarios are formulated:

- Find a use case in the scenario that specifies all possible 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 this use case in more detail
  - **Describe the entry condition**
  - **Describe the flow of events**
  - **Describe the exit condition**
  - **Describe exceptions**
  - **Describe special requirements (constraints, nonfunctional requirements)**

Use Cases

- A use case is a flow of events in the system, including interaction with actors
- It is initiated by an actor
- Each use case has a name
- Each use case has a termination condition
- Graphical Notation: An oval with the name of the use case



Use Case Model: **The set of all use cases specifying the complete functionality of the system**

Heuristics: How do I 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)**

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 from the scenario to participating actors
  - 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 Example: ReportEmergency Flow of Events

- The **FieldOfficer** activates the "Report Emergency" function of her terminal. FRIEND responds by presenting a form to the officer.
- The FieldOfficer fills 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, at which point, the **Dispatcher** is notified.
- 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 emergency report.
- The FieldOfficer receives the acknowledgment and the selected response.

Example of steps in formulating a use case

- Write down the exceptions:
  - **The FieldOfficer is notified immediately if the connection between her terminal and the central is lost.**
  - **The Dispatcher is notified immediately if the connection between any logged in FieldOfficer and the central is lost.**
- Identify and write down any special 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.**

Another Use Case Example: Allocate a Resource

- Actors:
  - *Field Supervisor:* **This is the official at the emergency site....**
  - *Resource Allocator:* **The Resource Allocator is responsible for the commitment and decommitment of the Resources managed by the FRIEND system. ...**
  - *Dispatcher:* **A Dispatcher enters, updates, and removes Emergency Incidents, Actions, and Requests in the system. The Dispatcher also closes Emergency Incidents.**
  - *Field Officer:* **Reports accidents from the Field**

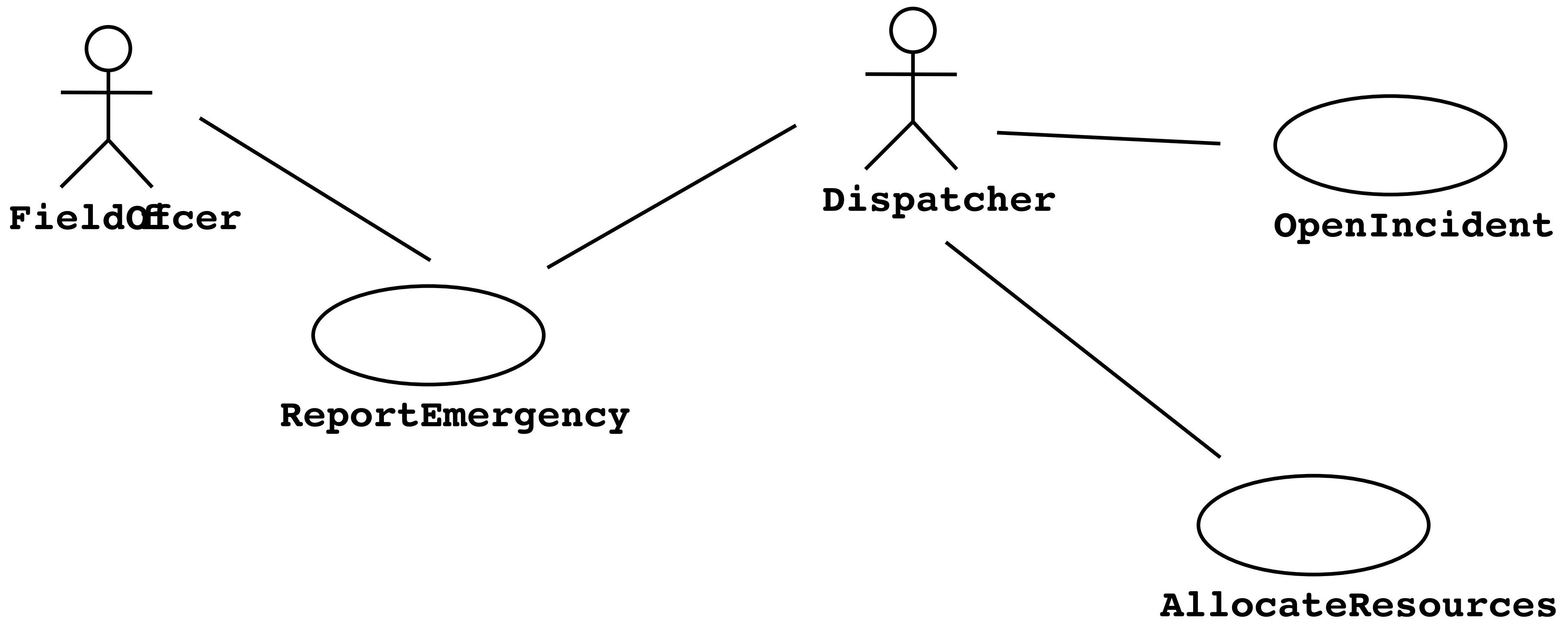
Another Use Case Example: Allocate a Resource

- *Use case name:* AllocateResources
- *Participating Actors:*
  - **Field Officer (Bob and Alice in the Scenario)**
  - **Dispatcher (John in the Scenario)**
  - **Resource Allocator**
  - **Field Supervisor**
- *Entry Condition*
  - **The Resource Allocator has selected an available resource.**
  - **The resource is currently not allocated**
- *Flow of Events*
  - **The Resource Allocator selects an Emergency Incident.**
  - **The Resource is committed to the Emergency Incident.**
- *Exit Condition*
  - **The use case terminates when the resource is committed.**
  - **The selected Resource is now unavailable to any other Emergency Incidents or Resource Requests.**
- *Special Requirements*
  - **The Field Supervisor is responsible for managing the Resources**

How to Specify a Use Case (Summary)

- Name of Use Case
- Actors
  - **Description of actors involved in use case**
- Entry condition
  - **Use a syntactic phrase such as “This use case starts when…”**
- Flow of Events
  - **Free form, informal natural language**
- Exit condition
  - **Start with “This use cases terminates when…”**
- Exceptions
  - **Describe what happens if things go wrong**
- Special Requirements
  - **List nonfunctional requirements and constraints**

Use Case Model for Incident Management



Another Use Case Example

Flow of Events

- The Bank Customer specifies a Account and provides credentials to the Bank proving that he is authorized to access the Bank Account
- The Bank Customer specifies the amount of money 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 Name **Withdraw Money Using ATM**

Participating Actor: Bank Customer

Entry condition:

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

Exit condition:

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

Flow of Events: A Request-Response Interaction between Actor and System

Actor steps	System steps
1.The Bank Customer inserts 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.

Event Flow: Use Indentation to show the Interaction between Actor and System

1.The Bank Customer inserts 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 Associations

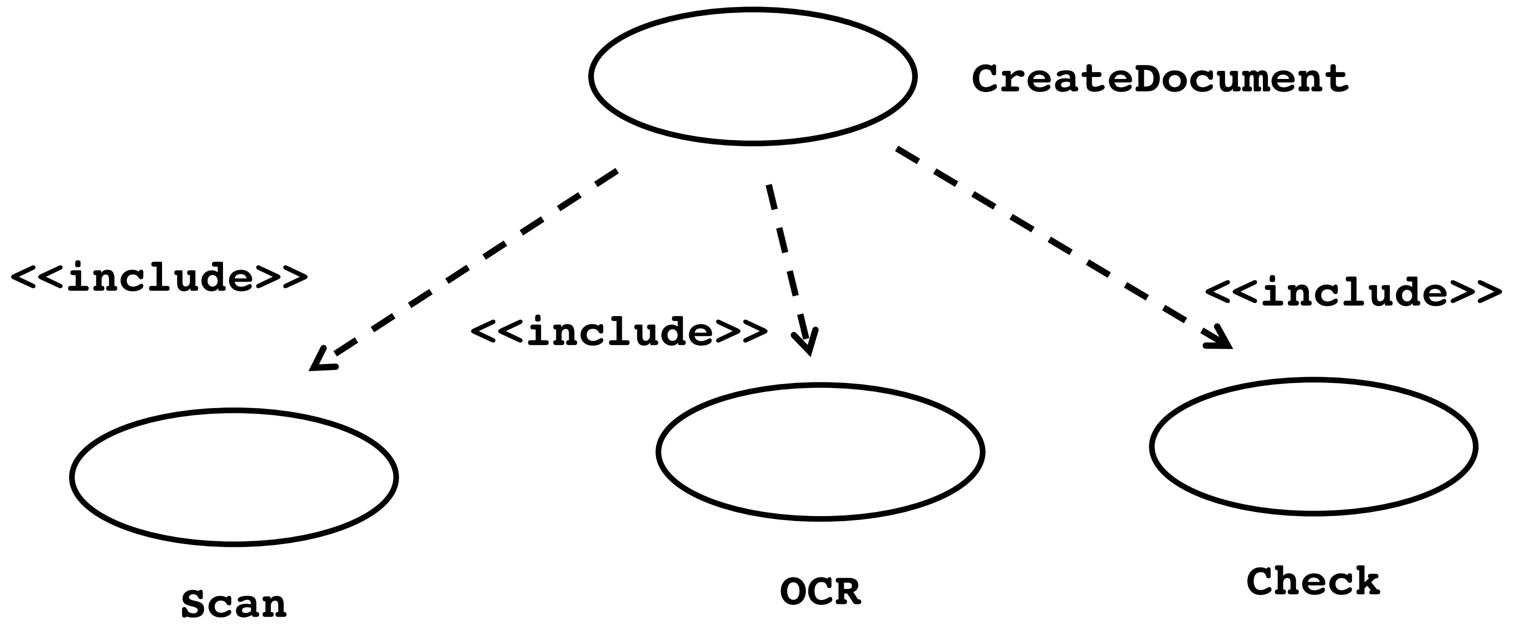
- Dependencies between use cases are represented with use case associations
- Associations are used to reduce complexity
  - Decompose a long use case into shorter ones
  - Separate alternate flows of events
  - Refine abstract use cases
- Types of use case associations
  - Includes
  - Extends
  - Generalization

Use Case Relationships

- A use case model consists of use cases and use case relationships
- Important types
  - **Dependencies**
    - **Include**
      - **A use case uses another use case ("functional decomposition")**
    - **Extend**
      - **A use case extends another use case**
  - **Generalization**
    - **An abstract use case has different specializations**

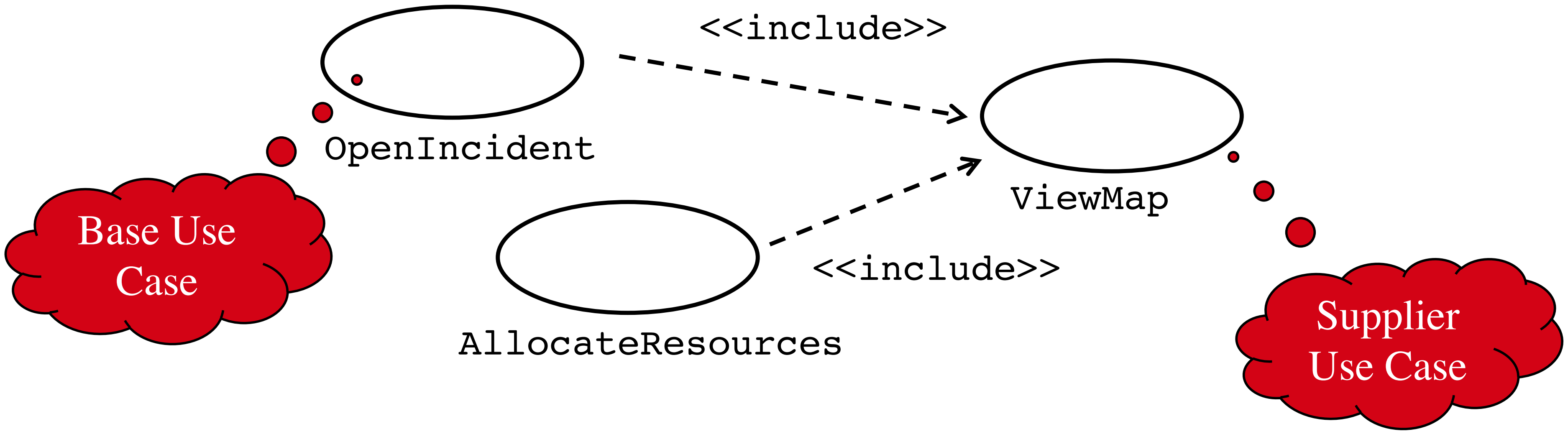
<<Include>>: Functional Decomposition

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



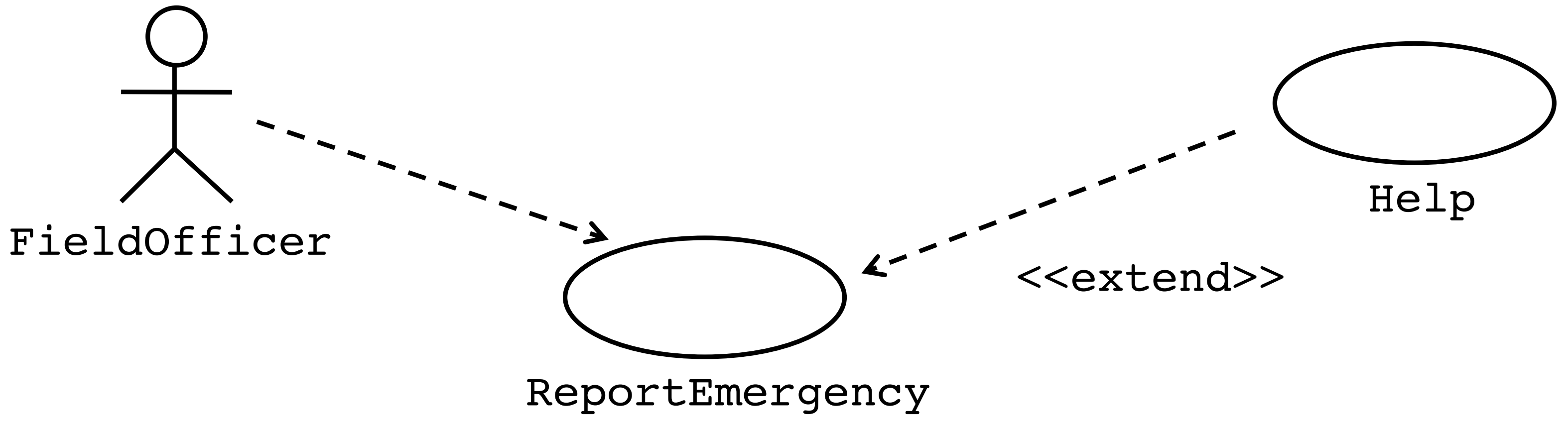
<<Include>>: Reuse of Existing Functionality

- **Problem**:
  - **There are already existing functions. How can we *reuse* them?**
- **Solution**:
  - **The *include relationship* from a use case A to a use case B indicates that an instance of the use case A performs all the behavior described in the use case B ("A delegates to B")**
- **Example**:
  - **The use case "ViewMap" describes behavior that can be used by the use case "OpenIncident" ("ViewMap" is factored out)**
- **Note: The base case cannot exist alone. It is always called with the supplier use case**



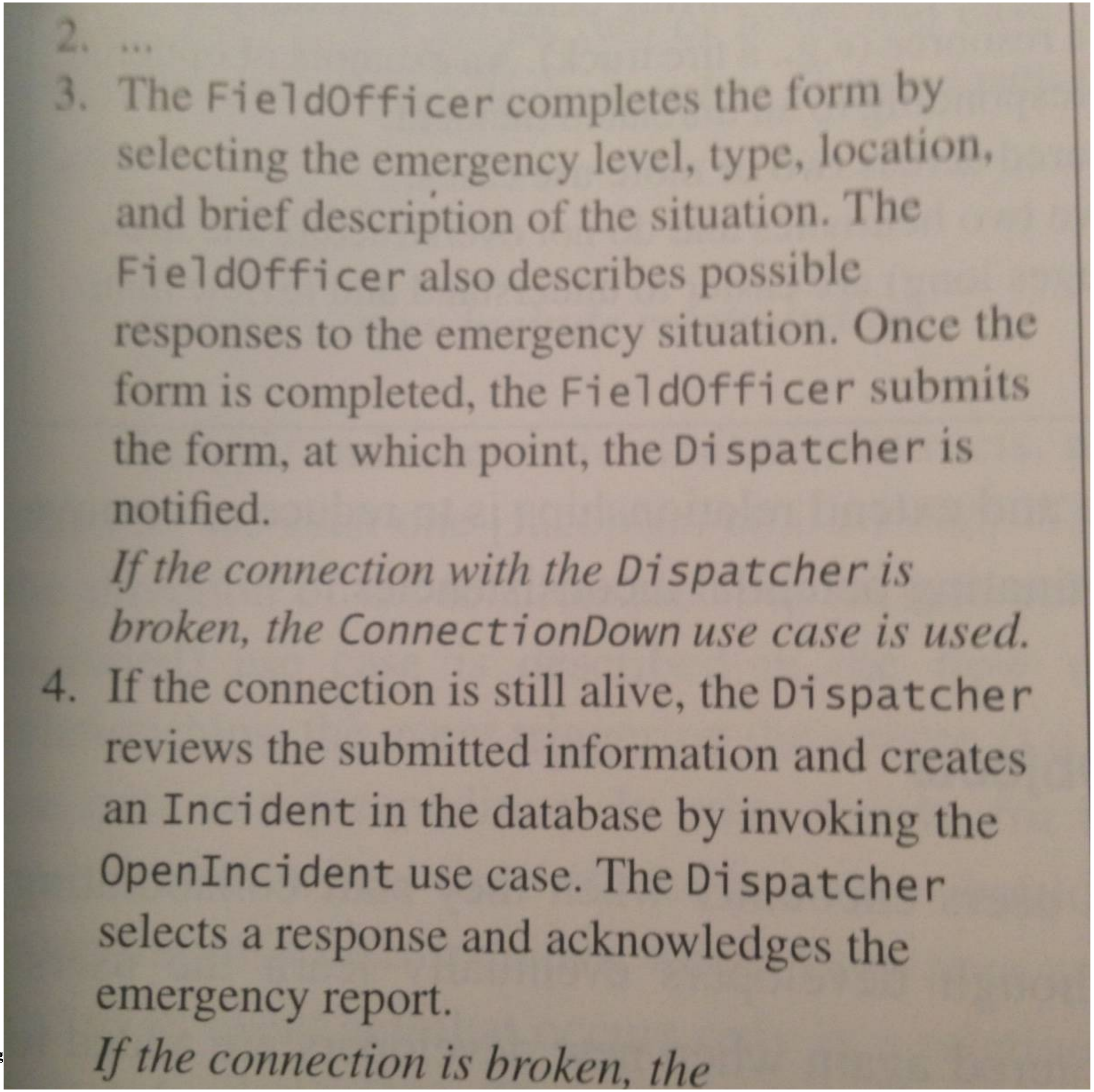
<Extend>> Relationship for Use Cases

- **Problem**:
  - **The functionality in the original problem statement needs to be extended.**
- **Solution**:
  - **An extend relationship from a use case A to a use case B indicates that use case A is an extension of use case B.**
- **Example**:
  - **The use case "ReportEmergency" is complete by itself , but can be extended by the use case "Help" for a specific scenario in which the user requires help**
- **Note**: In an extend relationship, the base use case can be executed without the use case extension

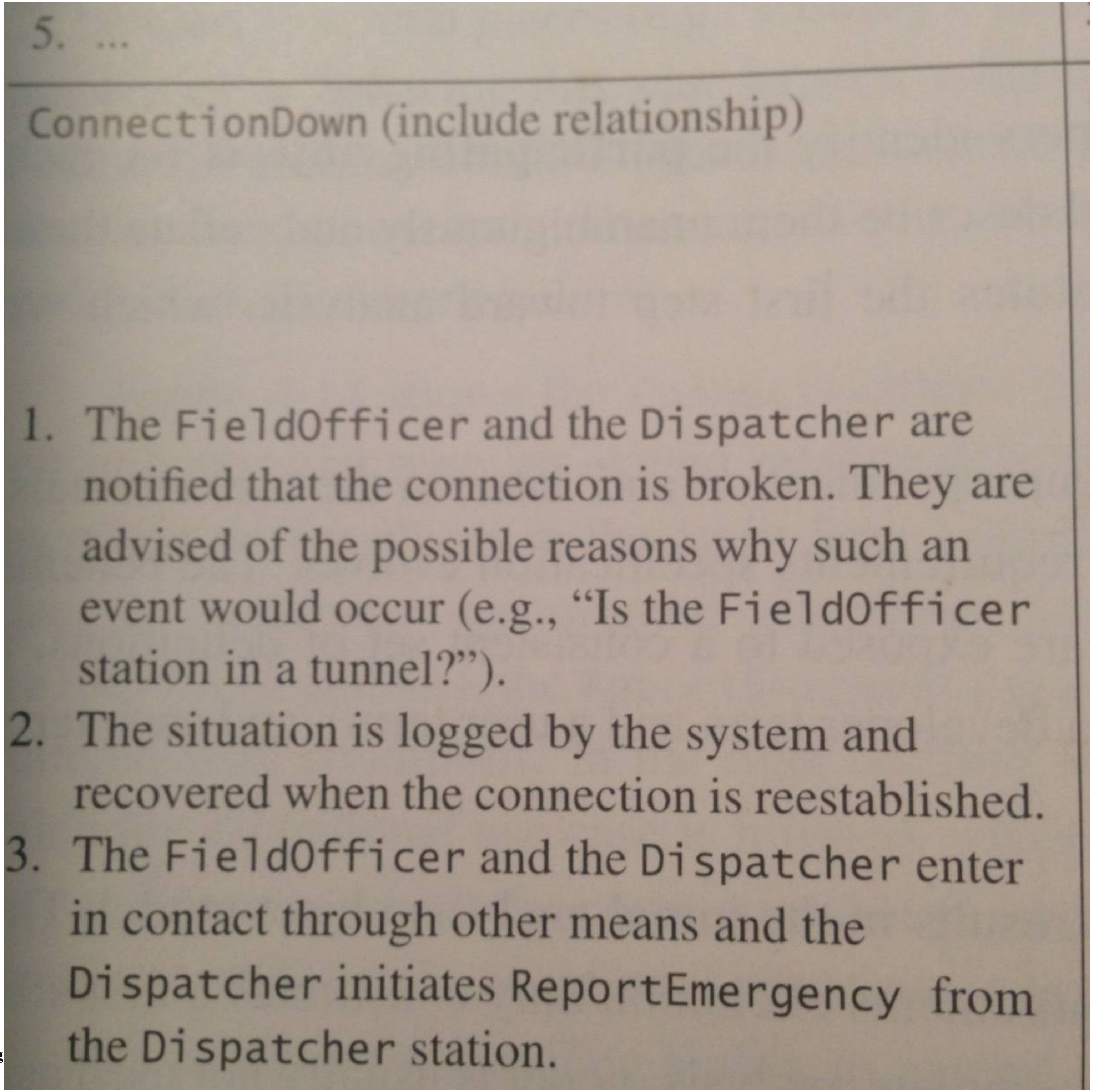


Extend versus include relationships

- **Extend**
  - **La condizione che attiva lo use case che estende è inserito nella entry condition dello use case che estende**
  - **Non occorre modificare lo /gli use case che vengono estesi**
  - **L'attivazione può avvenire in un punto qualsiasi del flusso di eventi dello use case base**
  - **Sono spessi situazioni eccezionali (failure, cancel, help…)**
- **Include**
  - **Ogni use case che include deve specificare dove viene invocato lo use case incluso**
  - **Funzioni comuni a più use case**
- **Esempio Fig. 4-14 libro**

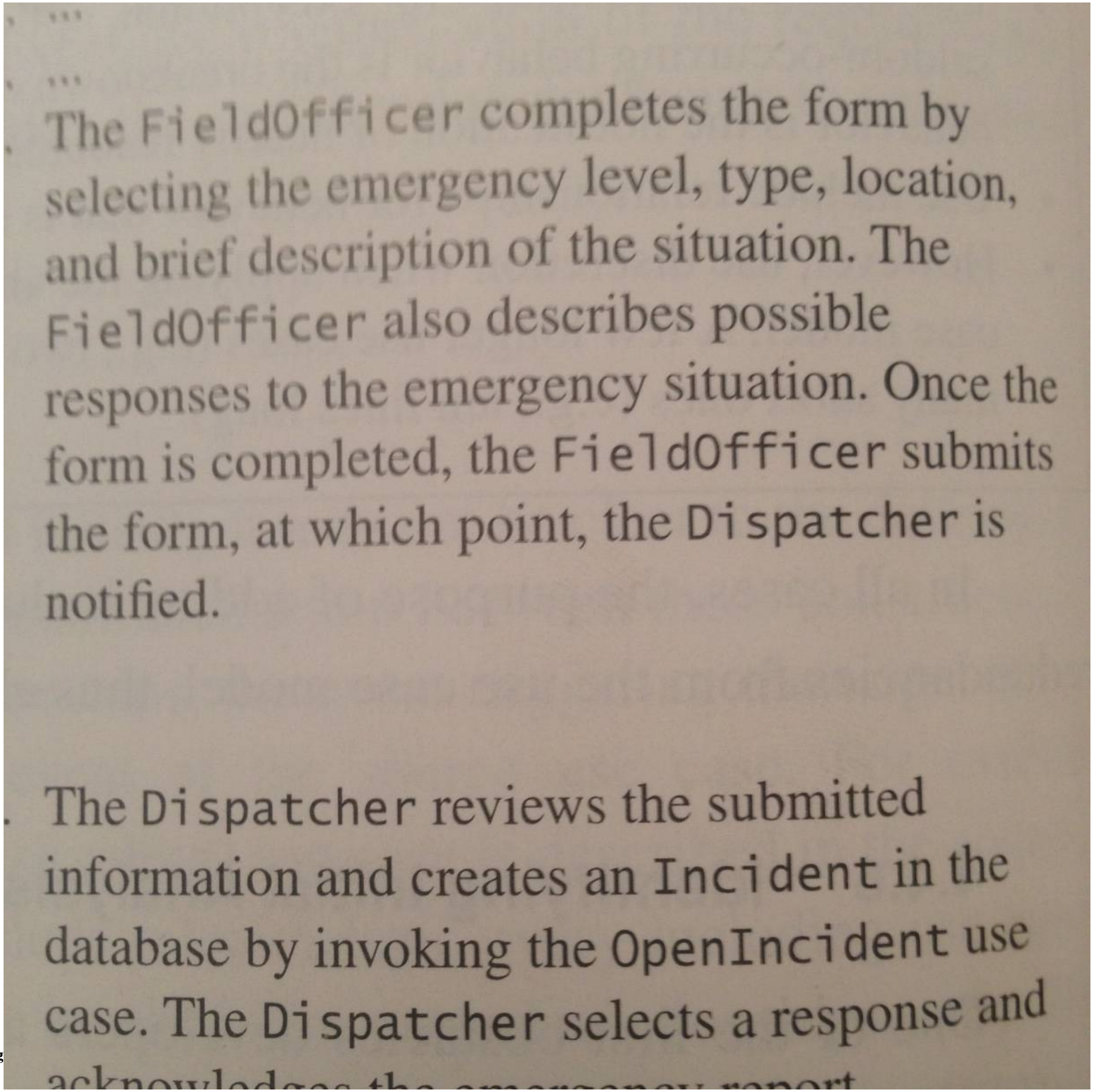


2. ...
3. The `FieldOfficer` completes the form by selecting the emergency level, type, location, and brief description of the situation. The `FieldOfficer` also describes possible responses to the emergency situation. Once the form is completed, the `FieldOfficer` submits the form, at which point, the `Dispatcher` is notified.
   If the connection with the `Dispatcher` is broken, the `ConnectionDown` use case is used.
4. If the connection is still alive, 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 emergency report.
   If the connection is broken, the


5. ...

ConnectionDown (include relationship)

1. The FieldOfficer and the Dispatcher are notified that the connection is broken. They are advised of the possible reasons why such an event would occur (e.g., “Is the FieldOfficer station in a tunnel?”).
2. The situation is logged by the system and recovered when the connection is reestablished.
3. The FieldOfficer and the Dispatcher enter in contact through other means and the Dispatcher initiates ReportEmergency from the Dispatcher station.

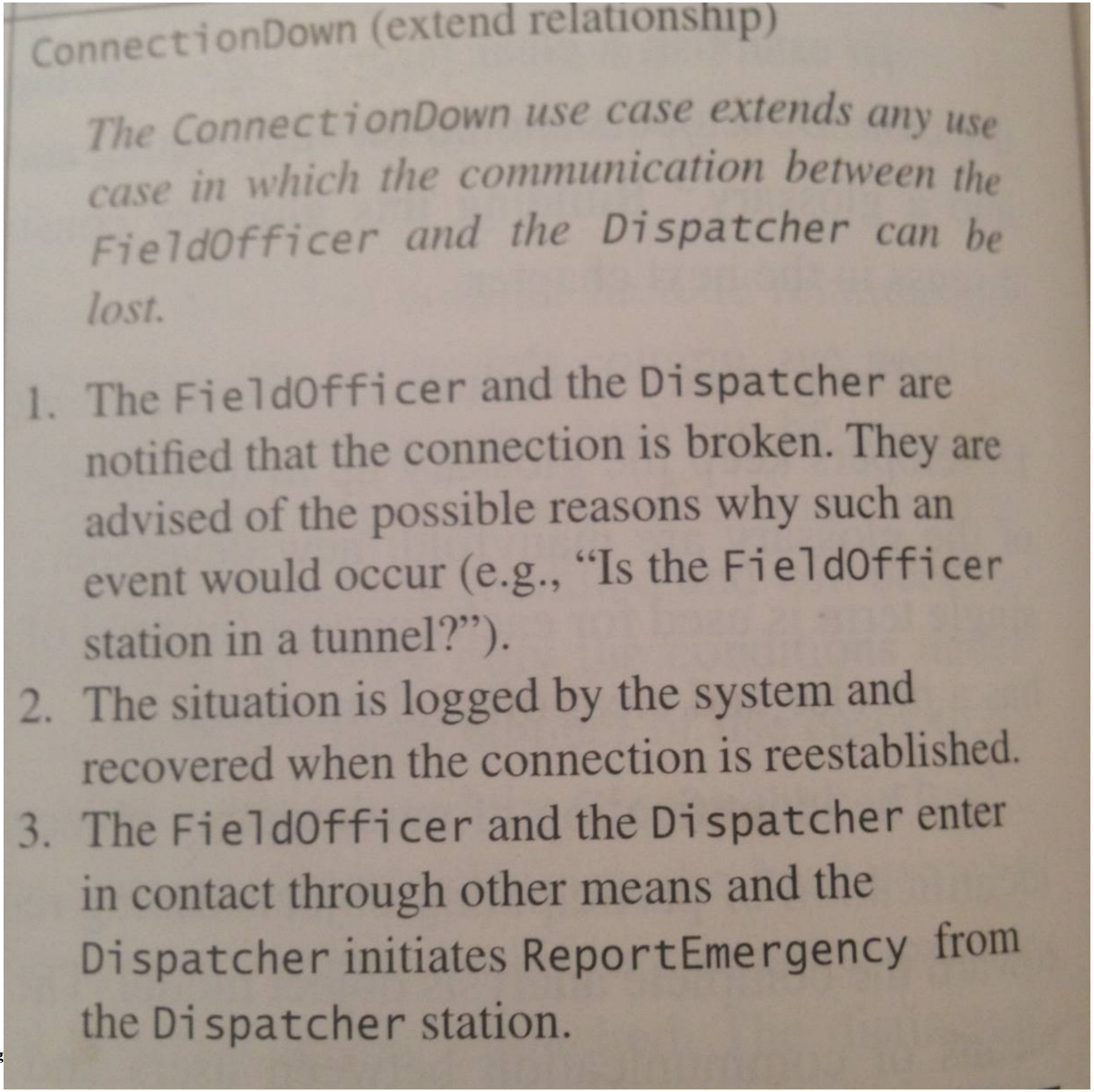


...

...

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, at which point, the `Dispatcher` is notified.

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 emergency report



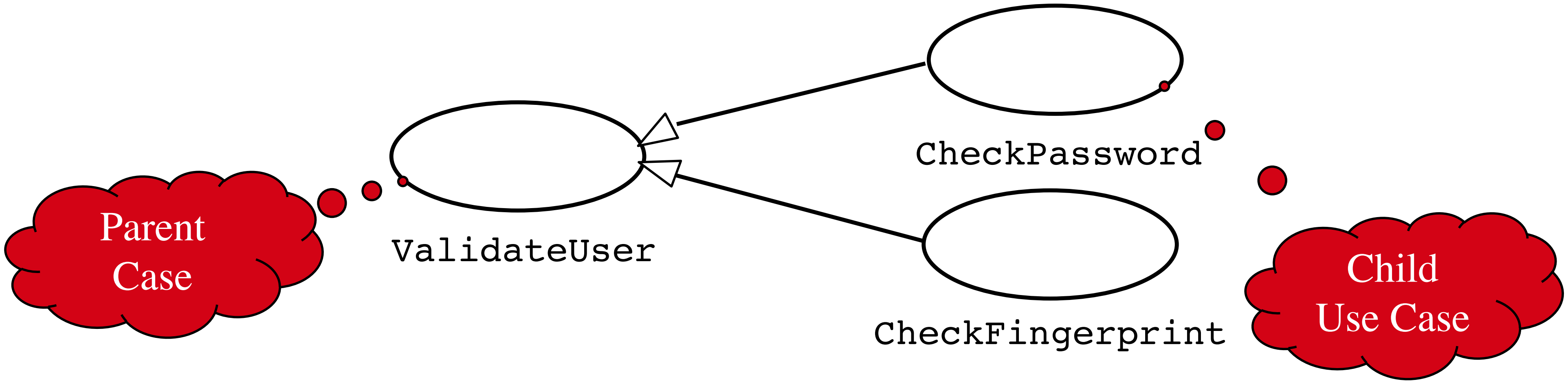
ConnectionDown (extend relationship)

The ConnectionDown *use case extends any use case in which the communication between the* FieldOfficer *and the* Dispatcher *can be lost.*

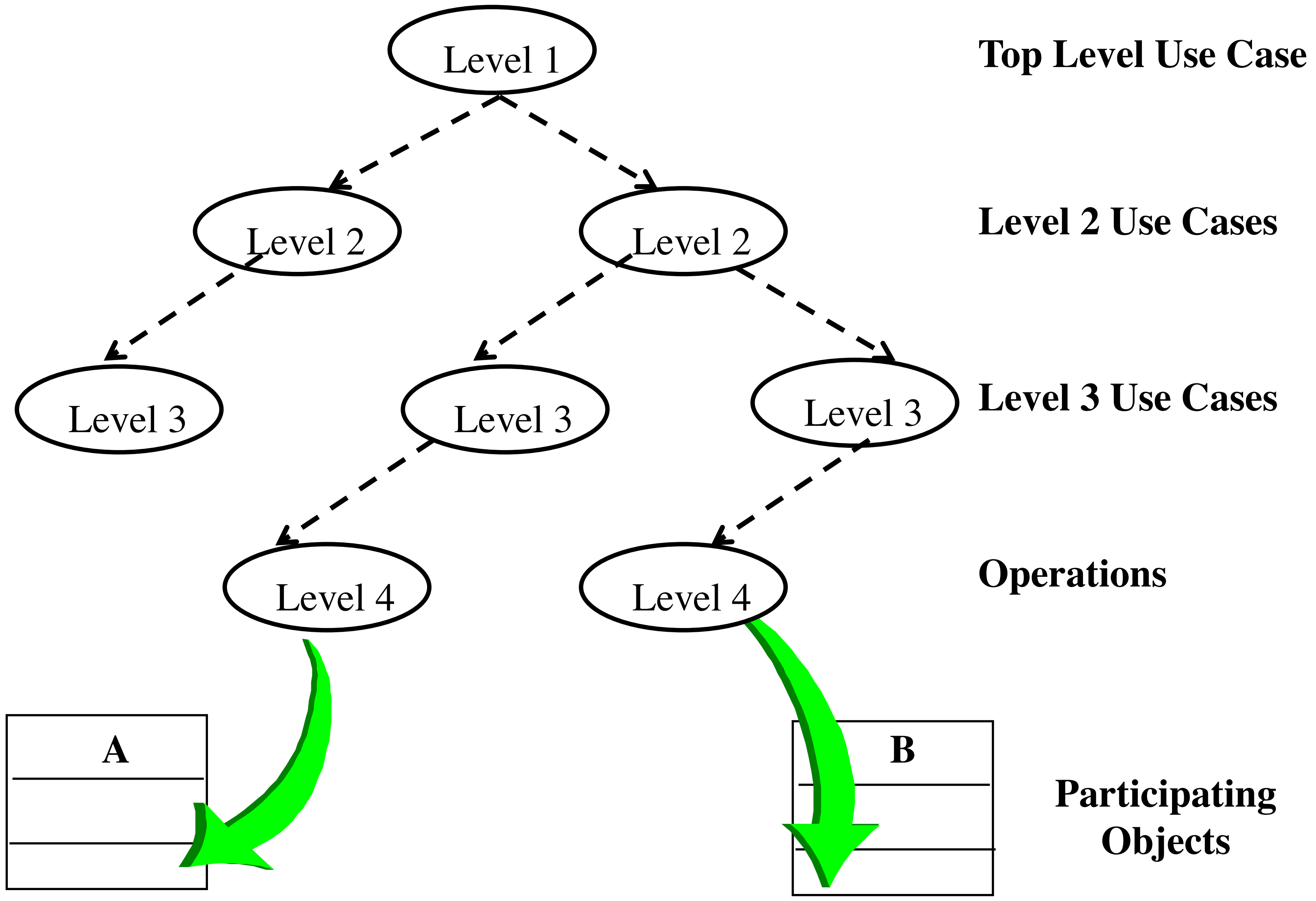
1. The FieldOfficer and the Dispatcher are notified that the connection is broken. They are advised of the possible reasons why such an event would occur (e.g., "Is the FieldOfficer station in a tunnel?").
2. The situation is logged by the system and recovered when the connection is reestablished.
3. The FieldOfficer and the Dispatcher enter in contact through other means and the Dispatcher initiates ReportEmergency from the Dispatcher station.

Generalization relationship in use cases

- **Problem**:
  - **You have common behavior among use cases and want to factor this out.**
- **Solution**:
  - **The generalization relation among use cases factors out common behavior. The child use cases inherit the behavior and meaning of the parent use case and add or override some behavior.**
- **Example**:
  - **Consider the use case "ValidateUser", responsible for verifying the identity of the user. The customer might require two realizations: "CheckPassword" and "CheckFingerprint"**



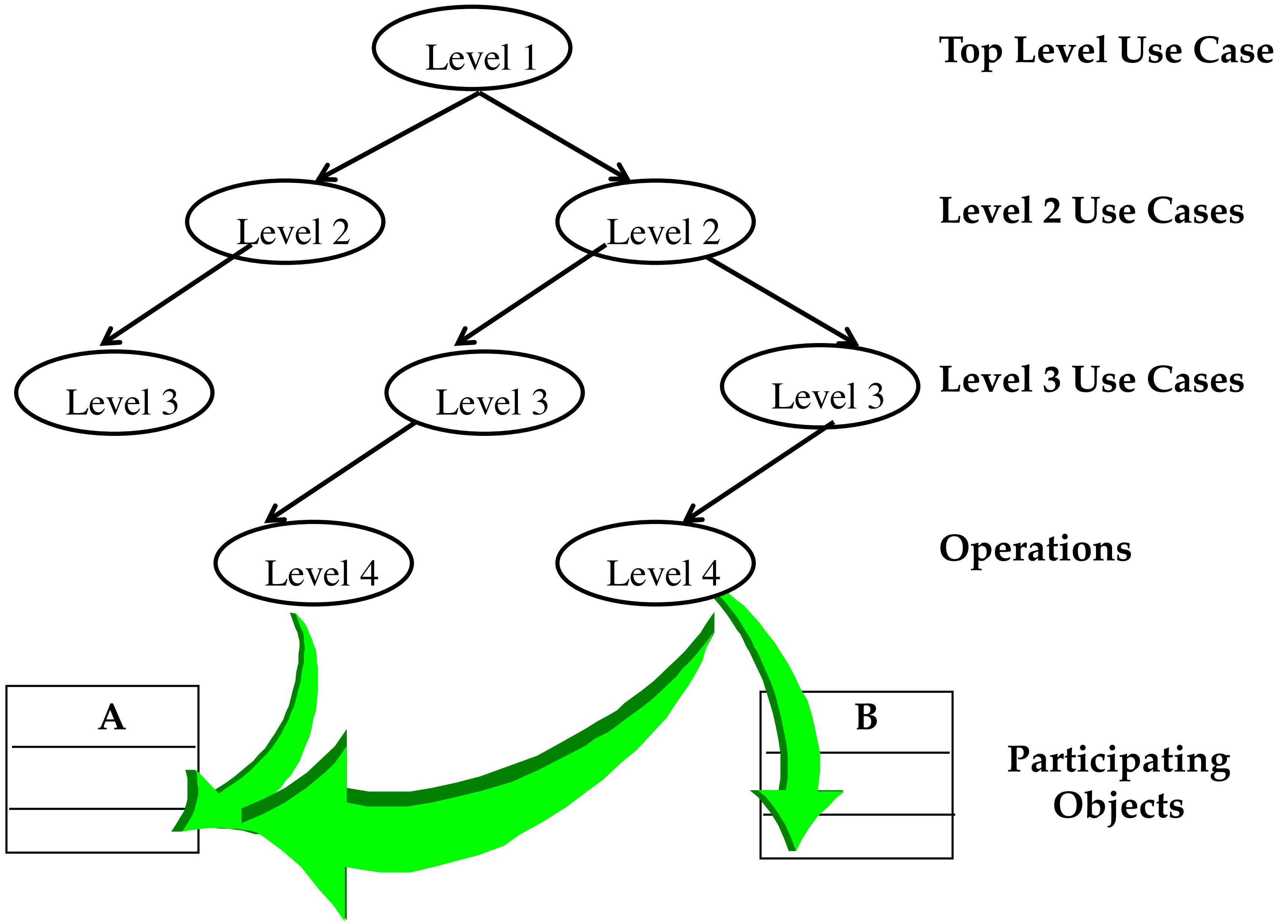
From Use Cases to Objects



Finding Participating Objects in Use Cases

- For any use case do the following
  - **Find terms that developers or users need to clarify in order to understand the flow of events**
    - **Always start with the user’s terms, then negotiate:**
      - **FieldOfficerStationBoundary or FieldOfficerStation?**
      - **IncidentBoundary or IncidentForm?**
      - **EOPControl or EOP?**
  - **Identify real world entities that the system needs to keep track of. Examples: FieldOfficer, Dispatcher, Resource**
  - **Identify real world procedures that the system needs to keep track of. Example: EmergencyOperationsPlan**
  - **Identify data sources or sinks. Example: Printer**
  - **Identify interface artifacts. Example: PoliceStation**
  - **Do textual analysis to find additional objects (Use Abott’s technique)**
  - **Model the flow of events with a sequence diagram**
- Build a data dictionary (or glossary)
- Cross-checking use cases and participating objects

Use Cases can be used by more than one object



Heuristics for cross-checking use cases and participating objects

- Which use cases create this object (ie.e, during which use cases are the values of the object attributes entered in the system)?
- Which actors can access this information?
- Which use cases modify and destroy this objct (i.e., which use cases edit or remove this information from the system)?
- Which actor can initiate this use case?
- Is this object needed (ie.e., is there at least one use case that depends on this information?)

Requirements

- Functional: describe the interactions between the system and its environment, independently from the implementation
- Non-functional: measurable/perceivable properties of the systems not directly related to functional aspects

Requirements specification

- Textual description of system behaviour
- Basic specification technique
- Most used in practice
- ISO/IEC/IEEE standard 29148:2011 (E)
- Should be accessible from the IEEE digital library (*https://ieeexplore.ieee.org/*)
- (slides partly based on 5.2 “Requirements fundamentals”)
- … I shall encourage you to read it !

Goal of a set of requirements

- ♦ enables an agreed understanding between stakeholders
  acquirers, users, customers, operators, suppliers
- ♦ is validated against real-world needs, can be implemented
- ♦ provides a basis of verifying designs and accepting solutions
- ♦ start with stakeholders intentions
- ♦ needs, goals, or objectives
- ♦ iterative process from stakeholders to system requirements

Well-formed requirements

- can be verified,
- has to be met or possessed by a system to solve a stakeholder problem or to achieve a stakeholder objective,
- is qualified by measurable conditions and bounded by constraints,
- defines the performance of the system when used by a specific stakeholder or the corresponding capability of the system, but not a capability of the user, operator, or other stakeholder.

What is a requirement, actually

- is a statement expressing a need and its associated constraints and conditions
- is written in natural language
  - **structural language, “semi-formal”**
- it comprises a subject, a verb, a complement
  - **subject of the requirement**
  - **what shall be done**

Some syntax example (1)

- **[Condition][Subject][Action][Object][Constraint]**

- When signal x is received [Condition], the system [Subject] shall set [Action] the signal x received bit [Object] within 2 seconds [Constraint].

Some syntax example (2)

- **[Condition][Subject][Action][Object][Constraint]**

- At sea state 1 [Condition], the Radar System [Subject] shall detect [Action] targets [Object] at ranges out to 100 nautical miles [Constraint].

Some syntax example (3)

- **[Subject][Action][Object][Constraint]**

- The invoice system [Subject], shall display [Action] pending customer invoices [Object] in ascending order in which invoices are to be paid [Constraint].

Important points (1)

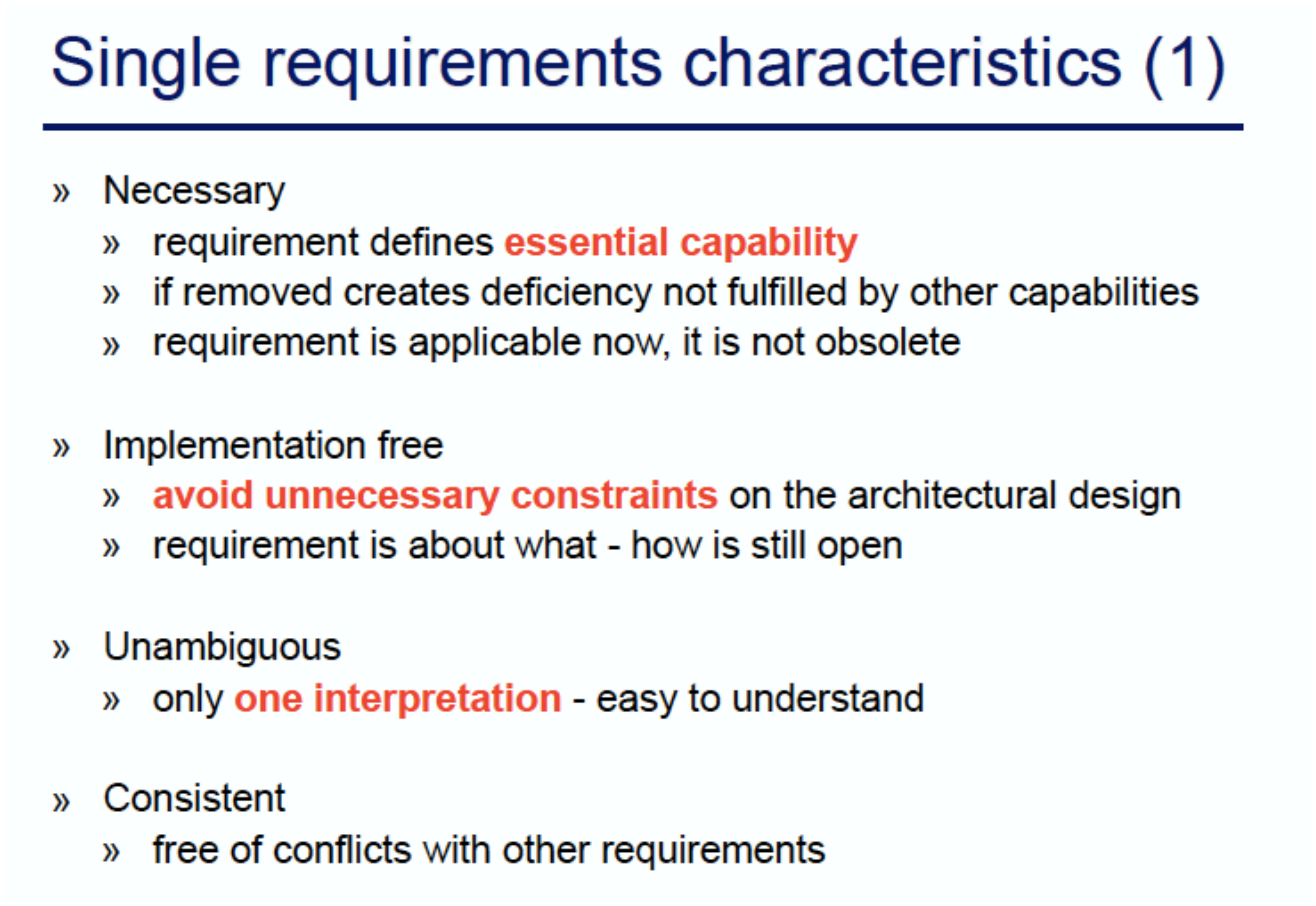
- Requirements:
- mandatory binding provisions and use ‘shall’ “deve”
- Preferences and goals
  - **desired, non-mandatory, or non-binding use ‘should’ “dovrebbe”**
- Suggestions or allowance
  - **non-mandatory, non-binding, use ‘may’ “può”**
- Non-requirements, such as descriptive text
  - **use verbs such as ‘are’, ‘is’, ‘was’**
- avoid ‘must’ to prevent confusion with a requirement

Important points (2)

- Use positive statements
  - **avoid negative statement as ‘shall not’**
- Use active voice
  - **avoid passive voice such as ‘shall be able to detect’**
  - **write ‘shall detect’**
- In general, all terms specific to requirements should be clearly defined and applied consistently throughout all requirements of the system

Examples of constraints

- interfaces to already existing systems, where the interfaces cannot be change
  - e.g. format, protocol, or content
- physical size limitations
  - e.g. a controller shall fit within a limited space in an airplane wing
- laws of particular country
- pre-existing technology platform
- user or operator capabilities and limitations
- …

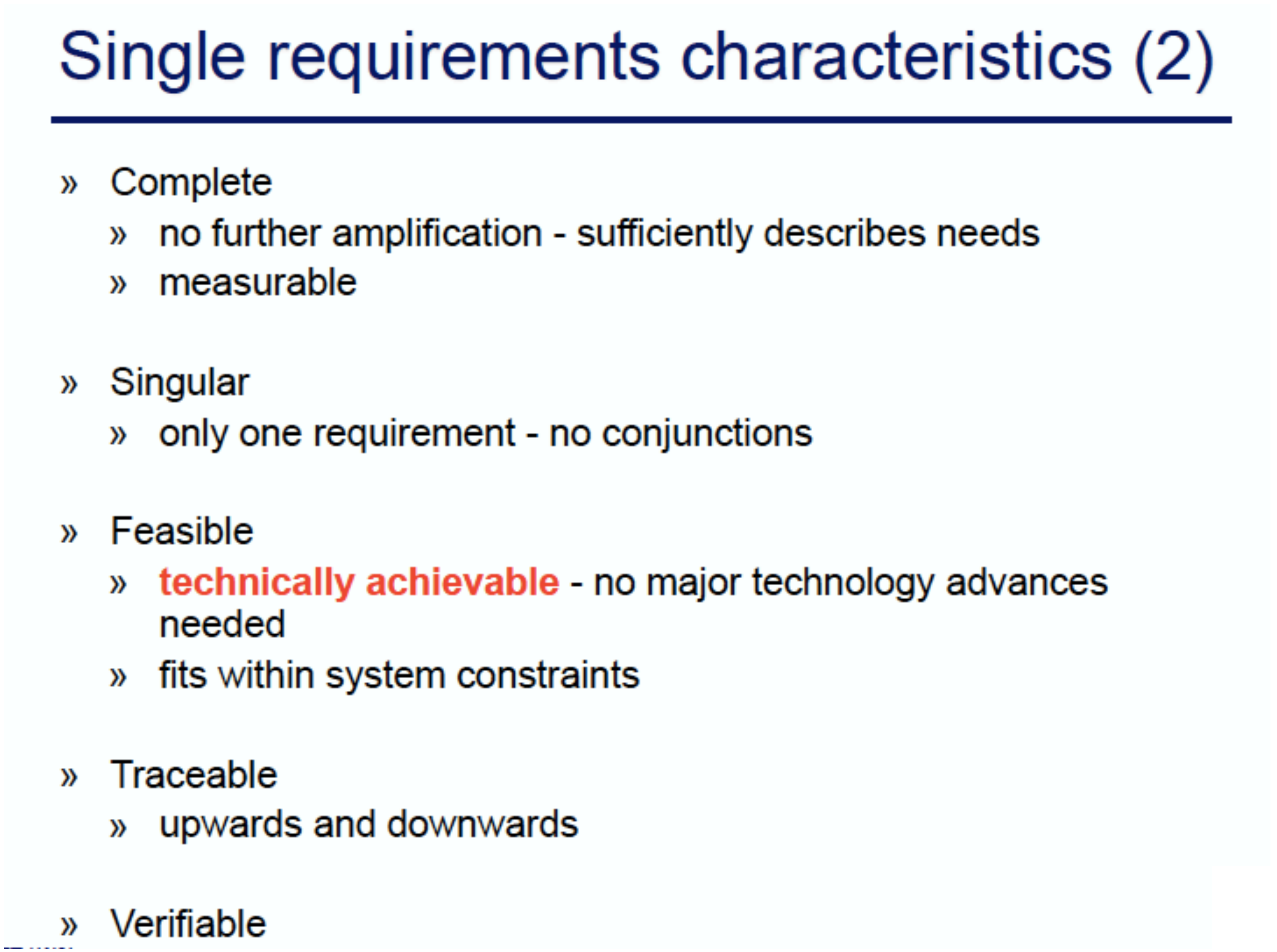


Single requirements characteristics (1)

- Necessary
  - requirement defines **essential capability**
  - if removed creates deficiency not fulfilled by other capabilities
  - requirement is applicable now, it is not obsolete
- Implementation free
  - **avoid unnecessary constraints** on the architectural design
  - requirement is about what - how is still open
- Unambiguous
  - only **one interpretation** - easy to understand
- Consistent
  - free of conflicts with other requirements

Non-conflicting

- » R1: When the water level exceeds V, the system shall shut-down the water pipe.
- » R2: When the fire sensor is activated, the system shall turn-on all water pipes.
- » What happen if my house has R1 and R2 and a fire is detected?

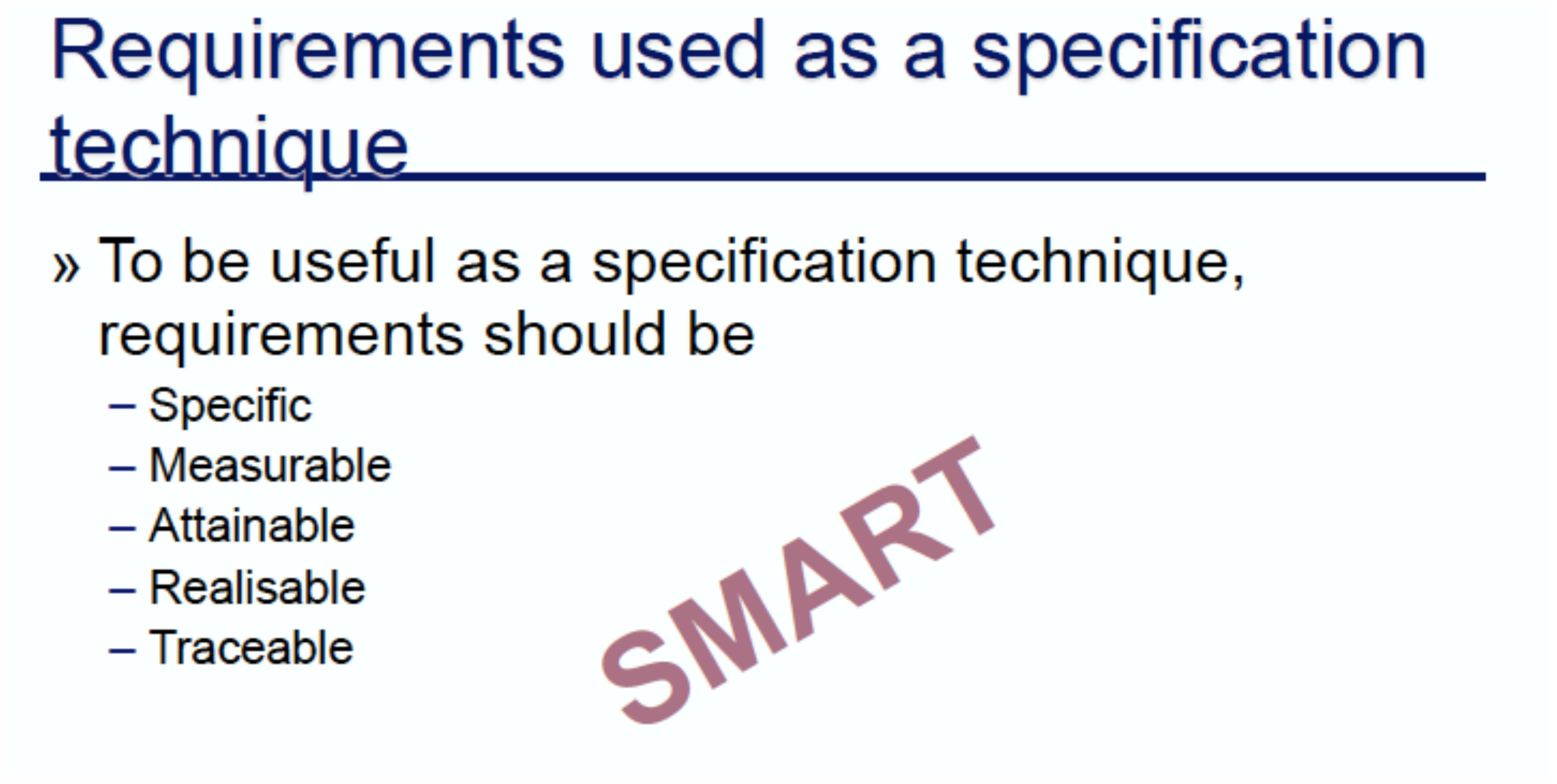


Single requirements characteristics (2)

- Complete
  - no further amplification - sufficiently describes needs
  - measurable
- Singular
  - only one requirement - no conjunctions
- Feasible
  - **technically achievable** - no major technology advances needed
  - fits within system constraints
- Traceable
  - upwards and downwards
- Verifiable

Set of requirements characteristics

- Complete
  - contains everything pertinent to the definition of the system
- Consistent
  - no conflicting requirements in the set
- Affordable
  - satisfied by a solution obtainable/feasible within life cycle constraints
- Bounded
  - remains within what is needed to satisfy user needs



Requirements used as a specification technique

» To be useful as a specification technique, requirements should be

- Specific
- Measurable
- Attainable
- Realisable
- Traceable

SMART

Traceability matrix

» Means of expressing traceability information

Require ment	**Design Elem.**	**Func**	**Test Case**
SR-28	Class Catalog	sort	7, 8
SR-44	Class Catalog	import	12, 13

Two popular techniques

What are their advantages and disadvantages?

Requirement	**Design element**		
	Class Catalog	**Class User**	**Class Book**
SR-28	*		
SR-44	*		
SR-62		*	*
SR-73			*

Unbounded or ambiguous terms (1) (to be avoided!)

- Superlatives (‘best’, ‘most’, …)
- Subjective language (‘user friendly’, ‘easy to use’, ‘cost effective’, …)
- Vague pronouns (‘it’, ‘this’, ‘that’, …)
  - When module A calls B **its** history memory file is updated
- Ambiguous adverbs and adjectives (‘significant’, ‘minimal’, …)
- Open-ended, non-verifiable terms (‘provide support’, ‘as a minimum’, ‘but not limited to’, …)

Unbounded or ambiguous terms (2) (to be avoided!)

- » Comparative phrases (‘better than, ‘higher quality’, …)
- » Loopholes (‘if possible’, ‘as appropriate’, ‘as applicable’, …)
- » Incomplete references
- » Negative statements (statement of capability not to be provided)

Type: functional vs. non-functional

» requirement, functional

A statement of some function or feature that should be implemented in a system *[Sommerville 2011].*

Type: functional vs. non-functional

» requirement, functional

A statement of some function or feature that should be implemented in a system *[Sommerville 2011]*.

We always build software for *somebody*

- **requirement, non-functional**

A statement of a constraint <…> that applies to a system *[Sommerville 2011]*.

Type: functional vs. non-functional

» requirement, functional

A statement of some function or feature that should be implemented in a system *[Sommerville 2011]*.

Functional (A) of non-functional (B) ?

The system sends an email to the customer when she places a new order.

functional

- **requirement, non-functional**

A statement of a constraint <…> that applies to a system *[Sommerville 2011]*.

Type: functional vs. non-functional

» requirement, functional

A statement of some <u>function</u> or <u>feature</u> that should be implemented in a system *[Sommerville 2011]*.

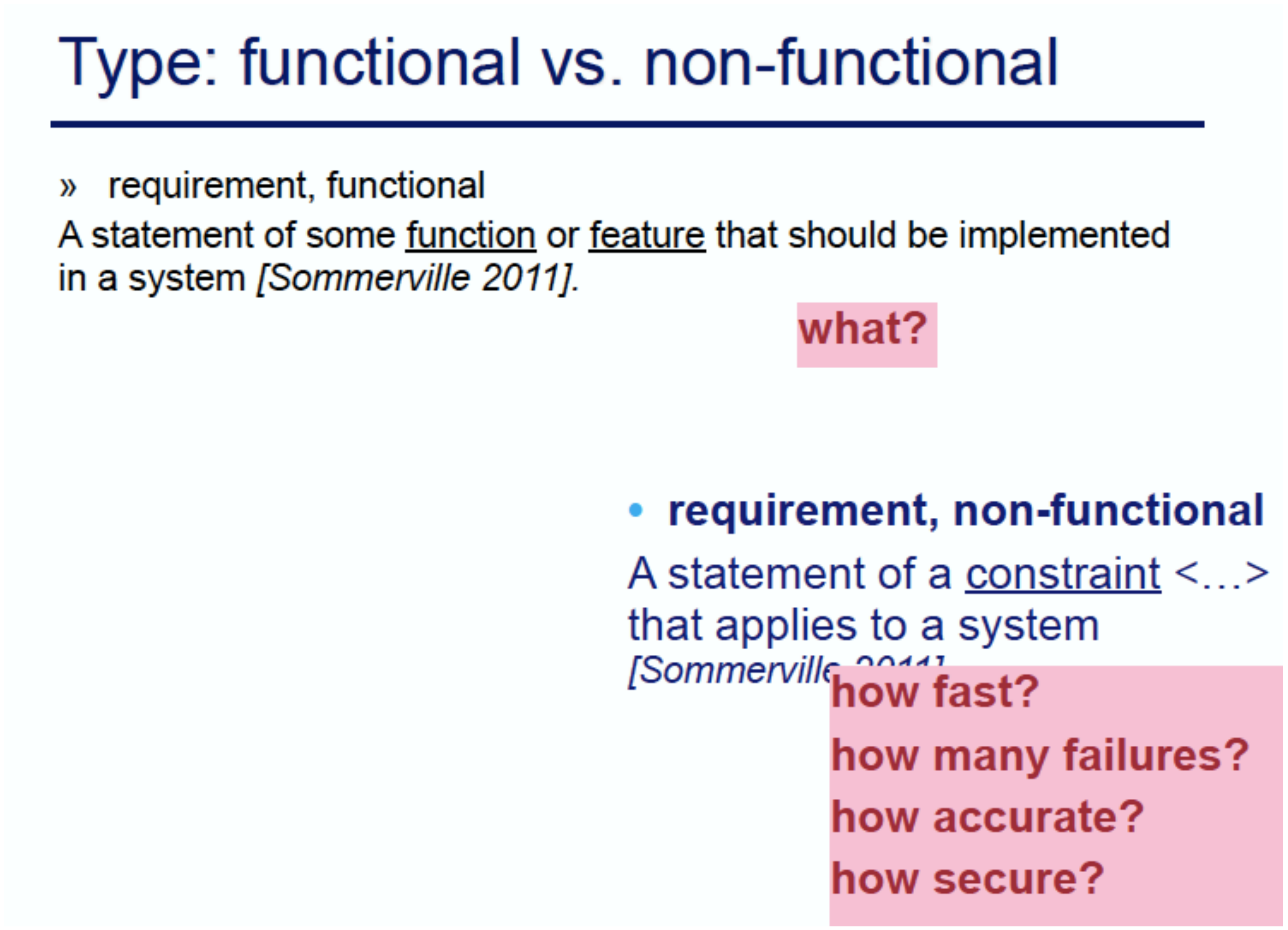
Functional (A) of non-functional (B)?

The mail should be sent not later than 12 hours after the order has been placed.

non-functional

- **requirement, non-functional**

A statement of a <u>constraint</u> <…> that applies to a system *[Sommerville 2011]*.



Type: functional vs. non-functional

» requirement, functional

A statement of some <u>function</u> or <u>feature</u> that should be implemented in a system *[Sommerville 2011]*.

what?

- **requirement, non-functional**

A statement of a <u>constraint</u> <…> that applies to a system *[Sommervill*

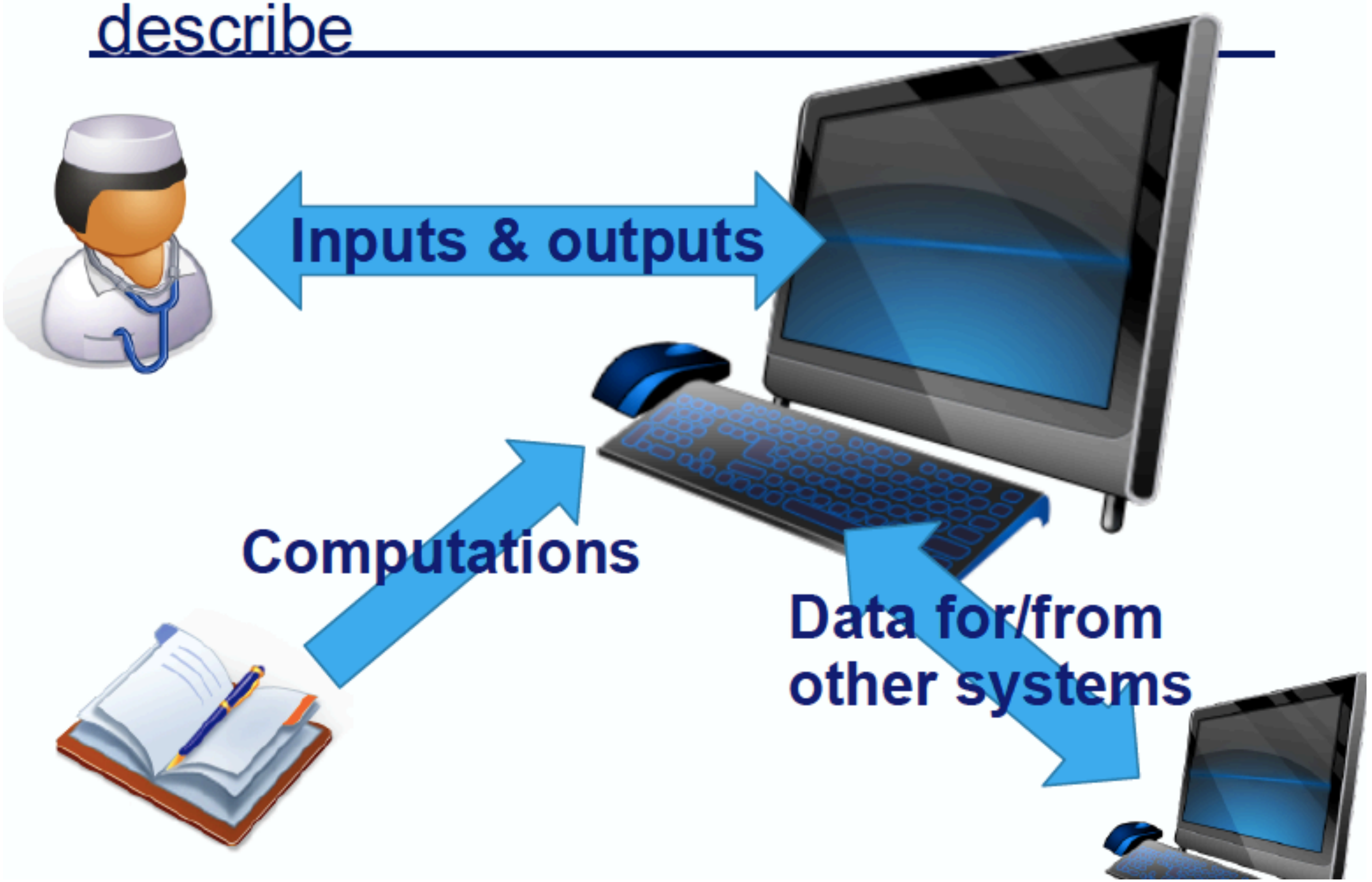
how fast?

how many failures?

how accurate?

how secure?

Functional requirements frequently describe



Non-functional requirements

» Non-functional requirement relates to quality attributes: e.g., **performance, learnability, availability**

» functional requirement: *"when the user presses the green button the Options dialog appears"*:
  – performance: how quickly the dialog appears;
  – availability: how often this function may fail, and how quickly it should be repaired;
  – learnability: how easy it is to learn this function.

Popular Quality Attributes (1)

- » reliability
  - – availability, fault tolerance, recoverability, ...
- » performance
  - – time, resource utilization
- » operability
  - – appropriateness recognizability, ease of use, use of interface aesthetics, technical accessibility, …

Popular Quality Attributes (2)

- security
  - confidentiality, integrity, authenticity, …
- compatibility
  - co-existence, interoperability
- maintainability
  - modularity, reusability, modifiability, testability, analyzability
- portability
  - adaptability, replaceability, installability

Non-functional requirements…

The system can connect to the scheduling system of the Human Resource department.

A	reliability	D	compatibility
B	performance	E	maintainability
C	operability	F	portability

Non-functional requirements…

The system can connect to the scheduling system of the Human Resource department.

A	reliability	D	compatibility
B	performance	E	maintainability
C	operability	F	portability

Answer: D (compatibility)

Be careful…

- » Sometimes the same idea may be expressed either as a **functional** or **non-functional** requirement.

- » The system shall ensure that data is protected from unauthorised access.
  - – Conventionally: non-functional requirement (security)
  - – Expressed as functional requirement:
    - The system shall include a user authorization procedure where users must identify themselves using a login name and password. Only users who are authorized in this way may access the system data

Ranking requirements

- » Limited resources, time, budget, …
- » Solution: check whether requirements are **realisable**
  - » can be implemented
- » Tips & tricks: prioritise requirements
  - – Must satisfy
  - – Should satisfy
  - – Could satisfy
  - – Would not satisfy [in this release]
  - ➢ MoSCoW

FURPS+: nonfunctional requirements categories

- **Usability** is the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.
    - **Conventions adopted by the user interface, the scope of online help, and the level of user documentation, …**
- **Reliability** is the ability of a system or component to perform its required functions under stated conditions for a specified period of time.
    - **Mean time to failure, ability to detect specified faults or to withstand specified security attacks, …**
    - **Recently 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 reliability,** *robustness*, **and** *safety*
- **Performance** requirements are concerned with quantifiable attributes of the system, such as **response time**, **throughput**, **availability**, and **accuracy.**
- **Supportability** requirements are concerned with the ease of changes to the system after deployment, including for example, **adaptability**, **maintainability**, and internationalization.
    - **The ISO 9126 standard on software quality replaces this category with two categories:** *maintainability* **and** *portability*

FURPS+: Pseudo requirements categories

- ***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).
- ***Legal requirements*** are concerned with licensing, regulation, and certification issues.

Identifying nonfunctional requirements (1)

Usability	What is the level of expertise of the user? What user interface standards are familiar to the user? What documentation should be provided to the user?
Reliability *(including robustness, safety, and security)*	How reliable, available, and robust should the system be? Is restarting the system acceptable in the event of a failure? How much data can the system loose? How should the system handle exceptions? Are there safety requirements of the system? Are there security requirements of the system ?
Performance	How responsive should the system be? Are any user tasks time critical? How many concurrent users should it support? How large is a typical data store for comparable systems? What is the worse latency that is acceptable to users?
Supportability *(including maintainability and portability*)	What are the foreseen extensions to the system? Who maintains the system? Are there plans to port the system to different software or hardware environments?

Identifying nonfunctional requirements (2)

Implementation	Are there constraints on the hardware platform? Are constraints imposed by the maintenance team? Are constraints imposed by the testing team?
Interface	Should the system interact with any existing systems? How are data exported/imported into the system? What standards in use by the client should be supported by the system?
Operation	Who manages the running system?
Packaging	Who installs the system? How many installations are foreseen? Are there time constraints on the installation?
Legal	How should the system be licensed? Are any liability issues associated with system failures? Are any royalties or licensing fees incurred by using specific algorithms or components?

Nonfunctional Requirements (Questions to overcome “Writers block”)

User interface and human factors

- What type of user will be using the system?
- Will more than one type of user be using the system?
- What training will be required for each type of user?
- Is it important that the system is easy to learn?
- Should users be protected from making errors?
- What input/output devices are available

Documentation

- What kind of documentation is required?
- What audience is to be addressed by each document?

Nonfunctional Requirements (2)

Hardware considerations

- What hardware is the proposed system to be used on?
- What are the characteristics of the target hardware, including memory size and auxiliary storage space?

Performance characteristics

- Are there speed, throughput, response time constraints on the system?
- Are there size or capacity constraints on the data to be processed by the system?

Error handling and extreme conditions

- How should the system respond to input errors?
- How should the system respond to extreme conditions?

Nonfunctional Requirements (3)

System interfacing

- Is input coming from systems outside the proposed system?
- Is output going to systems outside the proposed system?
- Are there restrictions on the format or medium that must be used for input or output?

Quality issues

- What are the requirements for reliability?
- Must the system trap faults?
- What is the time for restarting the system after a failure?
- Is there an acceptable downtime per 24-hour period?
- Is it important that the system be portable?

Nonfunctional Requirements (4)

System Modifications

- What parts of the system are likely to be modified?
- What sorts of modifications are expected?

Physical Environment

- Where will the target equipment operate?
- Is the target equipment in one or several locations?
- Will the environmental conditions be ordinary?

Security Issues

- Must access to data or the system be controlled?
- Is physical security an issue?

Nonfunctional Requirements (5)

Resources and Management Issues

- How often will the system be backed up?
- Who will be responsible for the back up?
- Who is responsible for system installation?
- Who will be responsible for system maintenance?

Managing requirements elicitation

- Negotiating Specifications with clients: Joint Application Design (JAD)
  - Different stakeholders (users, clients, developers) present their viewpoints, listen other viewpoint, negotiate, and come to a mutually acceptable solution
  - The requirements specification document is jointly developed by the different stakeholders
- Maintaining Traceability
  - Following the life cycle of a requirement
  - Useful to check that the system is complete, that the system complies with its requirements, to record the rationale behind the choices, and assess the impact of change
  - Cross-referencing among documents, models, and code artifacts and use simple tools (spreadsheets, word processors) to store dependences
  - Specialized tools: Rational RequisitePro, Telelogic DOORS

Requirements Analysis Document

l. Introduction
- 1.1 Purpose of the system
- 1.2 Scope of the system
- 1.3 Objectives and success criteria of the project
- 1.4 Definitions, acronyms, and abbreviations
- 1.5 References
- 1.6 Overview

2. Current system

3. Proposed system

4. Glossary

3. Proposed system
- 3.1 Overview
- 3.2 Functional requirements
- 3.3 Nonfunctional requirements
  - 3.3.1 Usability
  - 3.3.2 Reliability
  - 3.3.3 Performance
  - 3.3.4 Supportability
  - 3.3.5 Implementation
  - 3.3.6 Interface
  - 3.3.7 Packaging
  - 3.3.8 Legal
- 3.4 System models
  - 3.4.1 Scenarios
  - 3.4.2 Use case model
  - 3.4.3 *Object model*
  - 3.4.4 *Dynamic model*
  - 3.4.5 User interface-navigational paths and screen mock-ups

Requirements Elicitation (Summary)

- Requirements elicitation is to build a functional model of the system which will then be used during analysis to build an object model and a dynamic model
- Requirements Elicitation activities
  - **Identify actors**
  - **Identify scenarios**
  - **Identify use cases**
  - **Identify relationships among use cases**
  - **Refine use cases**
  - **Identify nonfunctional requirements**
  - **Identify participating objects**

Summary

- The requirements process consists of requirements elicitation and analysis.
- The requirements elicitation activity is different for:
  - **Greenfield Engineering, Reengineering, Interface Engineering**
- Scenarios:
  - **Great way to establish communication with client**
  - **Different types of scenarios: As-Is, visionary, evaluation and training**
  - **Use cases: Abstraction of scenarios**
- Pure functional decomposition is bad:
  - **Leads to unmaintainable code**
- Pure object identification is bad:
  - **May lead to wrong objects, wrong attributes, wrong methods**
- The key to successful analysis:
  - **Start with use cases and then find the participating objects**
  - **If somebody asks "What is this?", do not answer right away. Return the question or observe the end user: "What is it used for?"**