



POLITECNICO
MILANO 1863

Requirement Engineering

A possible scenario

- The municipality of Milano is telling you the following
 - The time needed to make decisions about granting permissions to build residential buildings in the city is too long
 - We want to develop a piece of software that helps us reducing this amount of time
- From where do we start from?
- How do we identify the most important aspects?
- How do we make sure that we have understood what our customers expect from us?



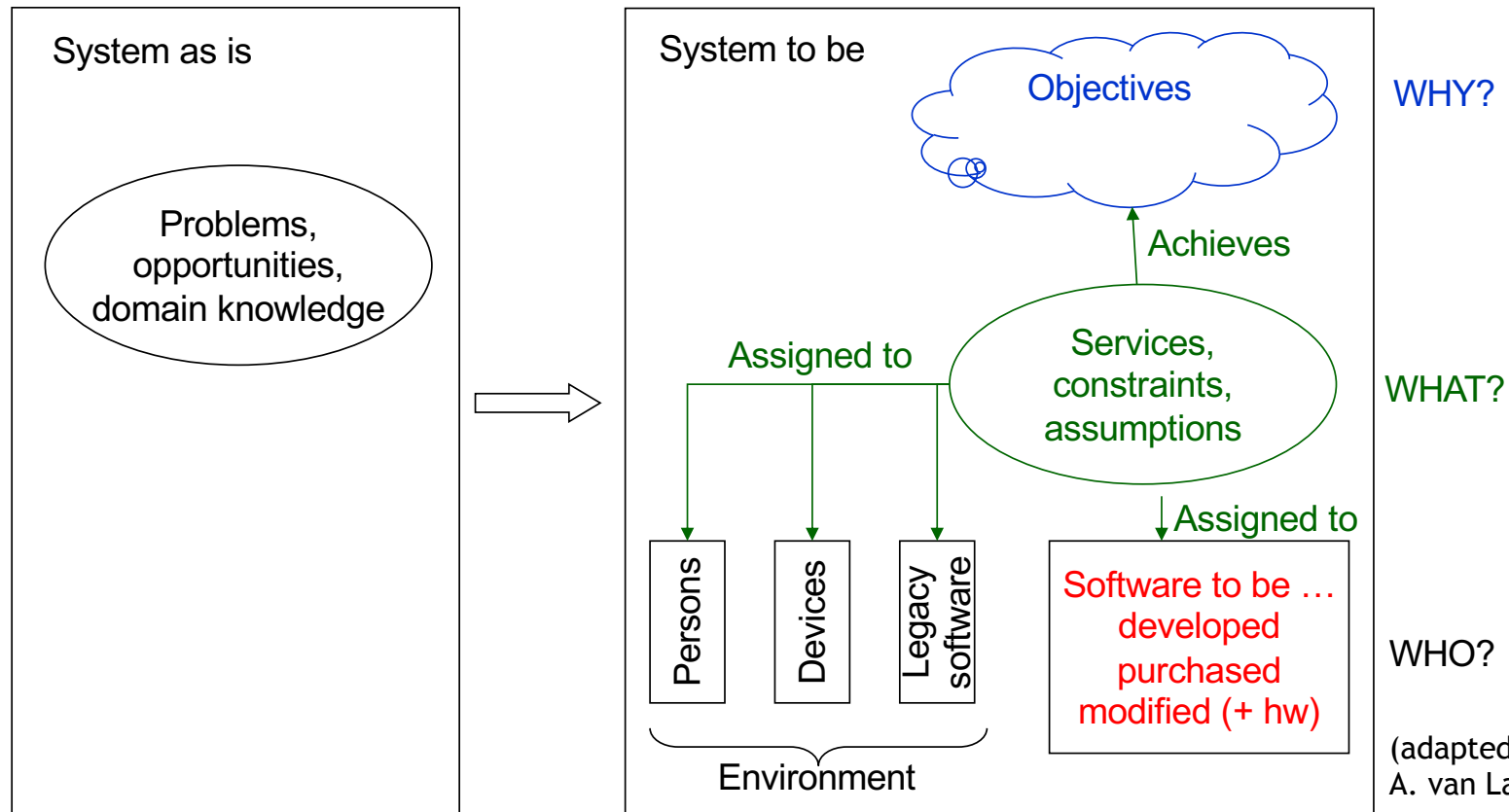
Requirements engineering: definition

- [Nuseibeh&Easterbrook '00]
 - The primary measure of success of a software system is the degree to which it meets the purpose for which it was intended
 - Software systems requirements engineering (RE) is the process of discovering that purpose, by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation
 - Important issues
 - *Identify stakeholders*
 - *Identify their needs*
 - *Produce documentation*
 - *Analyse, communicate, implement requirements*

Analyzing the system as is and the system to be



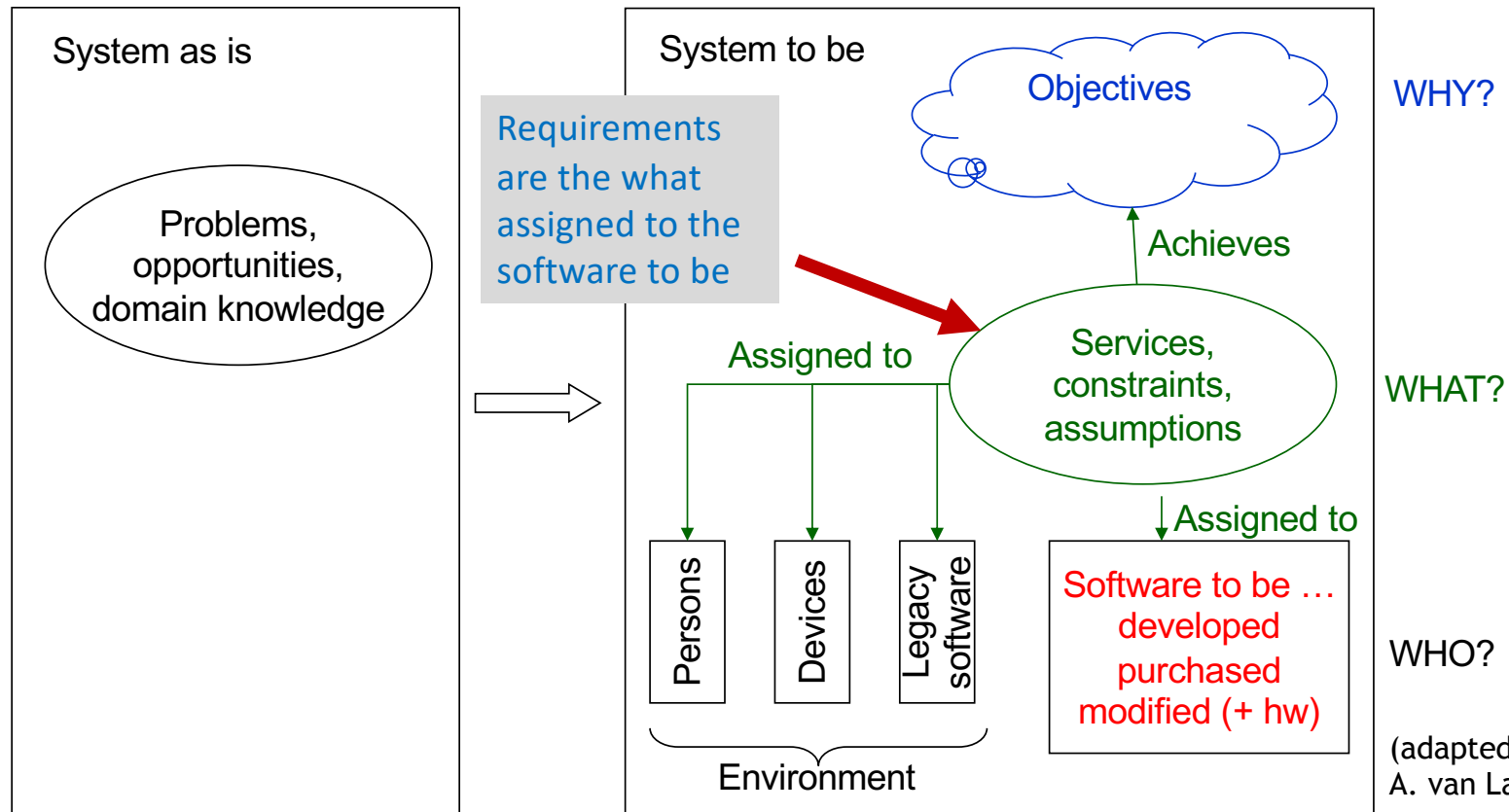
POLITECNICO
MILANO 1863



Analyzing the system as is and the system to be



POLITECNICO
MILANO 1863



(adapted from
A. van Lamsweerde)

Studying the interplay between the world and the machine

By M. Jackson and P. Zave, "Deriving Specifications from Requirements: an Example," 1995 17th International Conference on Software Engineering, Seattle, WA, USA, 1995, pp. 15-15, doi: 10.1145/225014.225016.

Example: ambulance dispatching system

- For every urgent call reporting an incident, an ambulance should arrive at the incident location within 14 mins
- For every urgent call, details about the incident are correctly encoded
- When an ambulance is dispatched, it will reach the incident location in the shortest possible time
- Accurate ambulance locations are known by GPS
- Ambulance crews correctly notify ambulance availability through a mobile data terminal

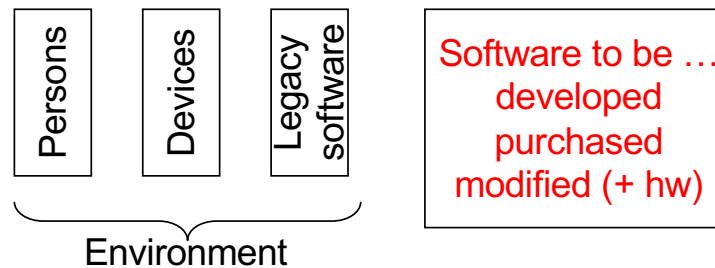


Examples of open questions

- Are you able to elicit requirements out of this description?
- **Possible questions**
 - Should the software system drive the ambulance?
 - Who or what is the "correctly encoding" details about incidents?
 - Do terminals exist already or not?
 - ...
- **More in general**
 - What is the boundary of the system? What is inside/outside? What is in-between?
 - How do we reason about these aspects in a systematic way?

The **World** and the **Machine**

(M. Jackson & P. Zave, 1995)



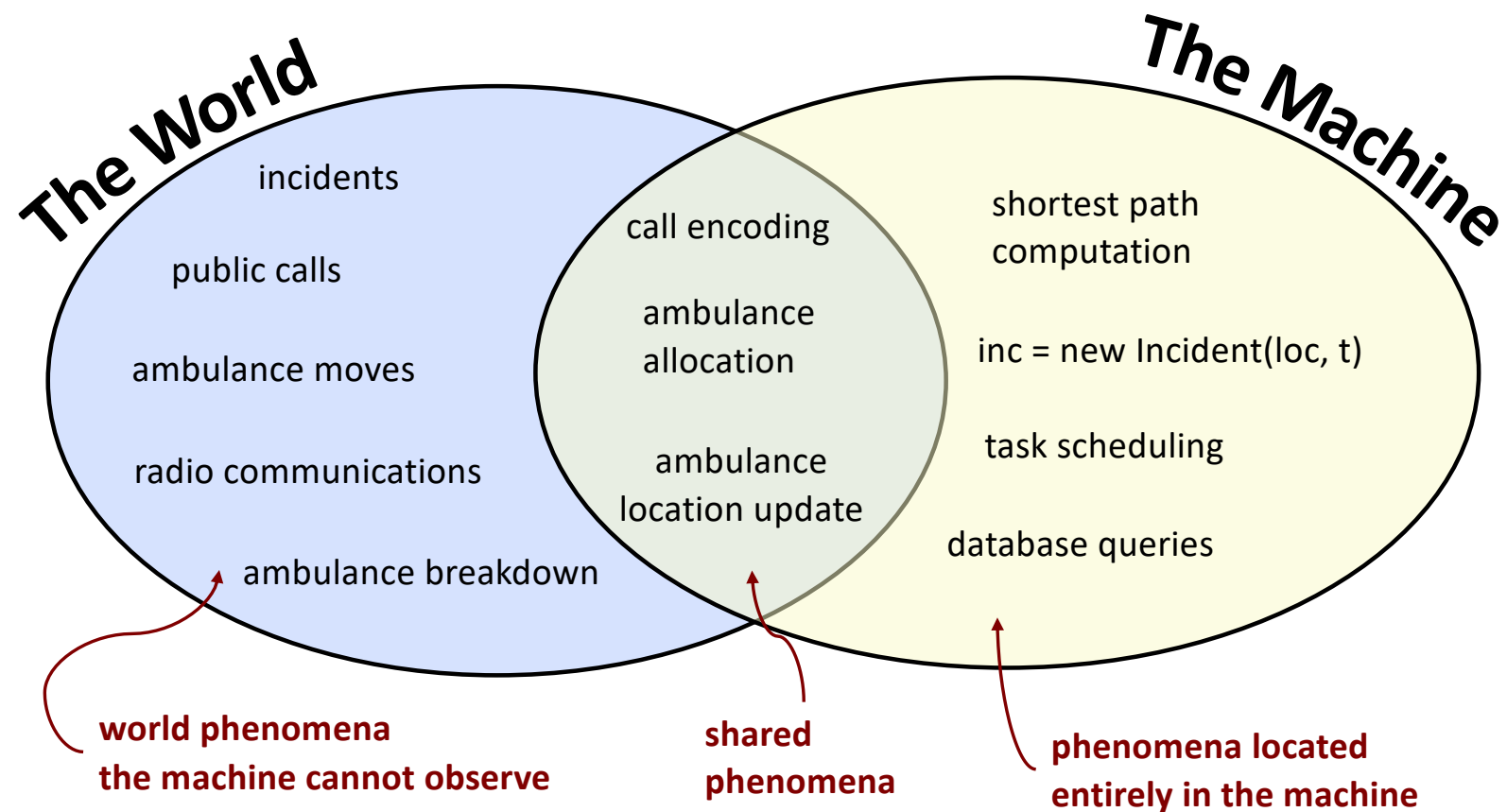
- Terminology

- The **machine** = the portion of system to be developed
typically, software-to-be + hardware
- The **world** (a.k.a. the environment) = the portion of the real-world affected by the machine

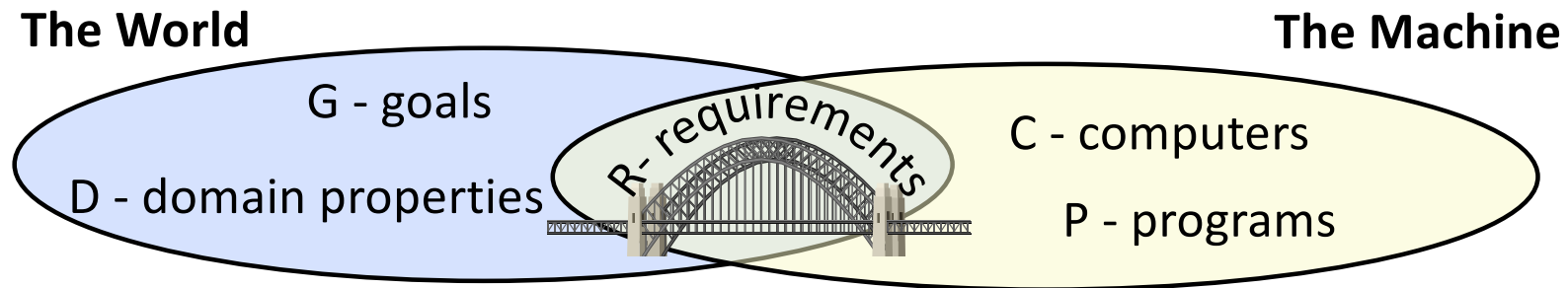
World and machine phenomena

- RE is concerned with **phenomena** occurring in the **world**
 - For an ambulance dispatching system:
 - Occurrences of incidents
 - Report of incidents by public calls
 - Encodings of calls' details into the dispatching software
 - Allocation of an ambulance
 - Arrival of an ambulance at the incident location
- As opposed to **phenomena** occurring inside the **machine**
 - For the same ambulance dispatching system:
 - the creation of a new object of class `Incident`
 - the update of a database entry
- **Requirement models are models of the world!**

The ambulance dispatching system



Goals, domain assumptions, requirements



- **Goals** → **prescriptive** assertions formulated in terms of world phenomena (not necessarily shared)
- **Domain properties/assumptions** → **descriptive** assertions **assumed to hold** in the world
- **Requirements** → **prescriptive** assertions formulated in terms of **shared** phenomena

Goals, domain assumptions, and requirements

- **Goal:**

- *For every urgent call reporting an incident, an ambulance should arrive at the incident scene within 14 minutes*

- **Domain assumptions:**

- *For every urgent call, details about the incident are correctly encoded*
- *When an ambulance is mobilized, it will reach the incident location in the shortest possible time*
- *Accurate ambulances' locations are known by GPS*
- *Ambulance crews correctly signal ambulance availability through mobile data terminals on board of ambulances*

- **Requirement:**

- *When a call reporting a new incident is encoded, the Automated Dispatching Software should mobilize the nearest available ambulance according to information available from the ambulances' GPS and mobile data terminals*



Completeness of Requirements

- Given the set of requirements **R**, goals **G** and domain assumptions **D**
- **R** is **complete** iff
 - **R** ensures satisfaction of **G** in the context of domain assumptions **D**

R and D \models G

- Analogy with program correctness: a Program **P** running on a particular Computer **C** is correct if it satisfies the Requirements **R**
 - **P** and **C** \models **R**
- **G** captures all the stakeholders' needs
- **D** represents valid properties/assumptions about the world

What can go wrong when defining G, R, D?

The A320 example



Example – Airbus A320

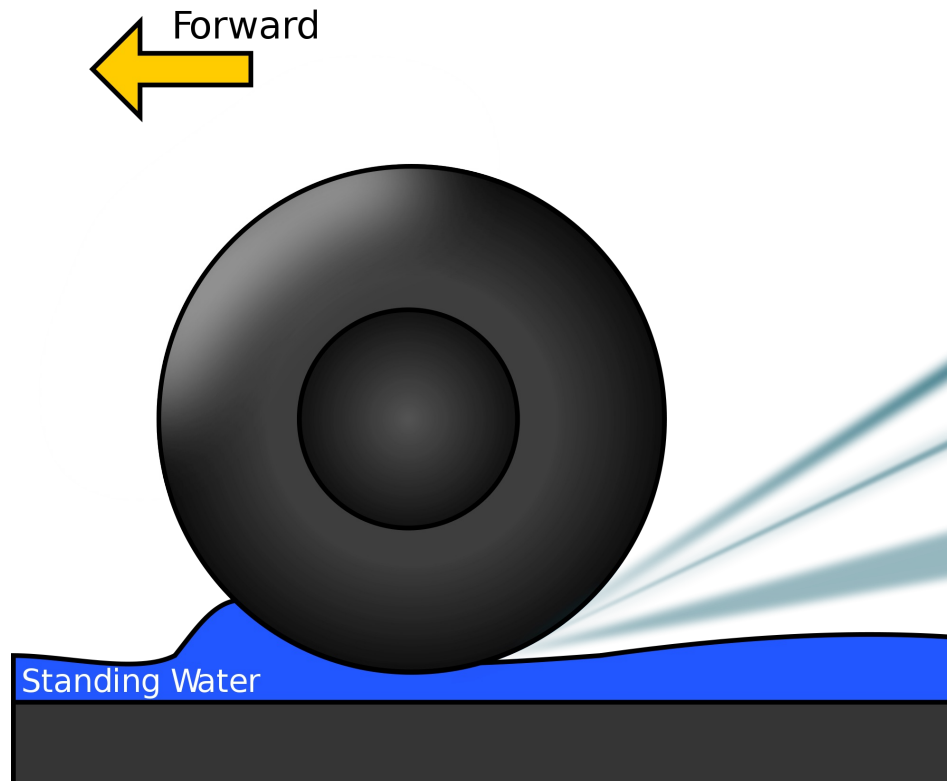
<https://aviation-safety.net/database/record.php?id=19930914-2>

- A Lufthansa Airbus on a flight from Frankfurt landed at Warsaw Airport in bad weather (rain and wind)
- On landing, the aircraft's software-controlled braking system did not deploy when activated by the flight crew and it was about 9 seconds before the braking system activated
- There was insufficient runway remaining to stop the plane and the aircraft ran into a grass embankment
- Two people were killed and 54 injured
- Several causes:
 - Human errors
 - Software errors (braking control system)



Lufthansa Airbus A.320-211 D-AIPN at Frankfurt 27 May 1990 , © Werner Fischdick

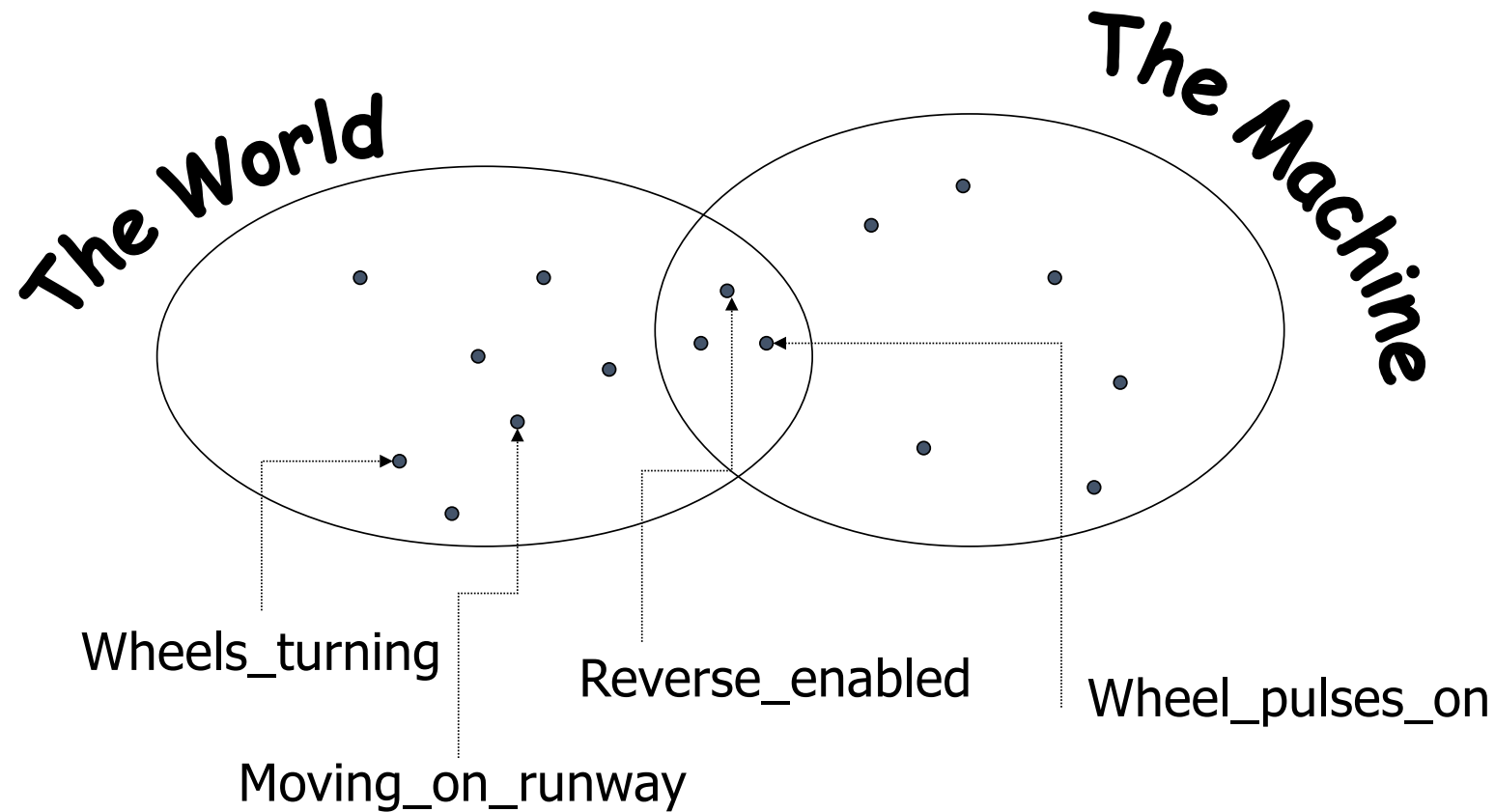
Ever heard of hydroplaning?



Domain assumptions D
do not apply to reality,
the correctness argument
is bogus!

Incorrect domain
assumptions
lead to disasters!

Example – Airbus A320 Braking Logic





Goal, domain assumptions, and requirement

- **Goal G:**
 - “Reverse thrust shall be enabled if and only if the aircraft is moving on the runway”
- **Domain Assumptions D:**
 - Wheel pulses on if and only if wheels turning
 - Wheels turning if and only if moving on runway
- **Requirements R:**
 - Reverse thrust enabled if and only if wheel pulses on
- Verification: R and D \models G applies!
- ... but D are not valid assumptions!!!
- **Invalid domain assumptions -> Warsaw accident**

Formulating and classifying requirements

Working with examples

Formulating requirements

- Examples of candidate requirements
 - “The system shall allow users to reserve taxis”
 - “The system has to provide a feedback in 5 seconds”
 - “The system should never allow non-registered users to see the list of other users willing to share a taxi”
 - “The system should be available 24/7”
 - “The system should guarantee that the reserved taxi picks the user up”
 - “The system should be implemented in Java”
 - “The search for the available taxi should be implemented in class Controller”

Functional requirements

- Examples of candidate requirements
 - “The system shall allow users to reserve taxis”
 - “The system has to provide a feedback in 5 seconds”
 - “The system should never allow non-registered users to see the list of other users willing to share a taxi”
 - “The system should be available 24/7”
 - “The system should guarantee that the reserved taxi picks the user up”
 - “The system should be implemented in Java”
 - “The search for the available taxi should be implemented in class Controller”

Unfeasible (from the perspective of the software to be) functional requirement!

- Examples of candidate requirements
 - “The system shall allow users to reserve taxis”
 - “The system has to provide a feedback in 5 seconds”
 - “The system should never allow non-registered users to see the list of other users willing to share a taxi”
 - “The system should be available 24/7”
 - **“The system should guarantee that the reserved taxi picks the user up”**
 - “The system should be implemented in Java”
 - “The search for the available taxi should be implemented in class Controller”

Non-functional requirements

- Examples of candidate requirements
 - “The system shall allow users to reserve taxis”
 - “The system has to provide a feedback in 5 seconds”
 - “The system should never allow non-registered users to see the list of other users willing to share a taxi”
 - “The system should be available 24/7”
 - “The system should guarantee that the reserved taxi picks the user up”
 - “The system should be implemented in Java”
 - “The search for the available taxi should be implemented in class Controller”

Technical requirements

- Examples of candidate requirements
 - “The system shall allow users to reserve taxis”
 - “The system has to provide a feedback in 5 seconds”
 - “The system should never allow non-registered users to see the list of other users willing to share a taxi”
 - “The system should be available 24/7”
 - “The system should guarantee that the reserved taxi picks the user up”
 - “The system should be implemented in Java”
 - “The search for the available taxi should be implemented in class Controller”

Do we really need this as a requirement?

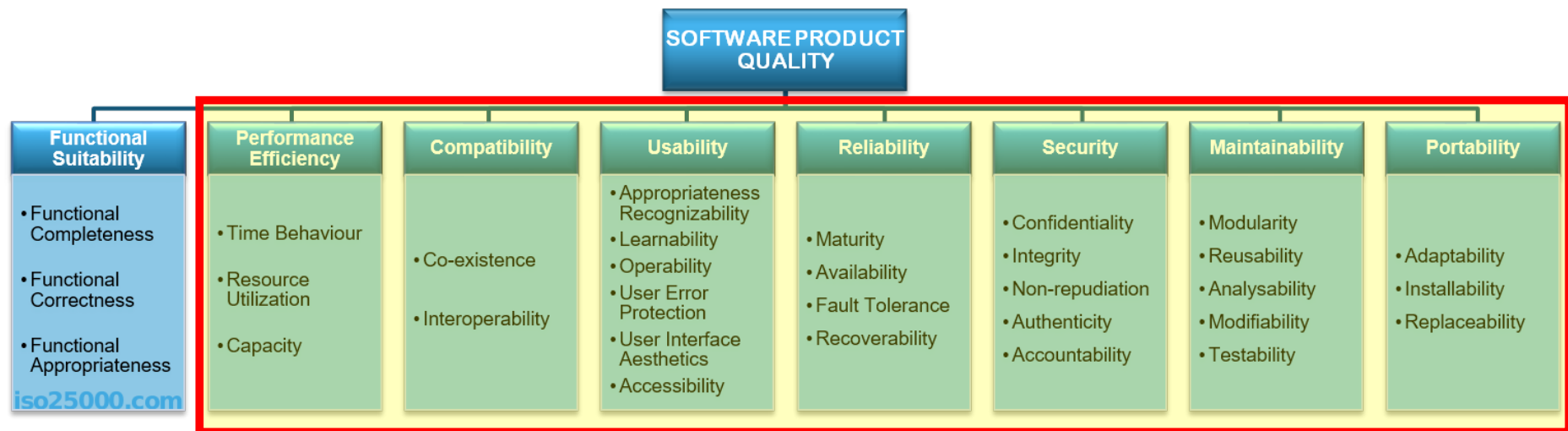


Types of requirements

- **Functional** Reqs: describe the **interactions** between the **system** and its **environment** (independent from implementation)
 - Examples:
 - “The word processor shall allow users to search for strings in the text”
 - “The system shall allow users to reserve taxis”
 - Are the main (functional) goals the software has to realize
- **Non-functional** Reqs (NFRs): further characterization of user-visible aspects of the system not directly related to functions
 - Examples:
 - “The response time must be less than 1 second”
 - “The server must be available 24 hours a day”
- **Constraints** (or technical requirements): imposed by the customer or the environment in which the system operates
 - “The implementation language must be Java”
 - “The credit card payment system must be able to be dynamically invoked by other systems relying on it”

NFRs and product qualities

- NFRs predicate over **external** non-functional qualities
 - Qualities must be measurable through **metrics**





Characteristics of NFRs

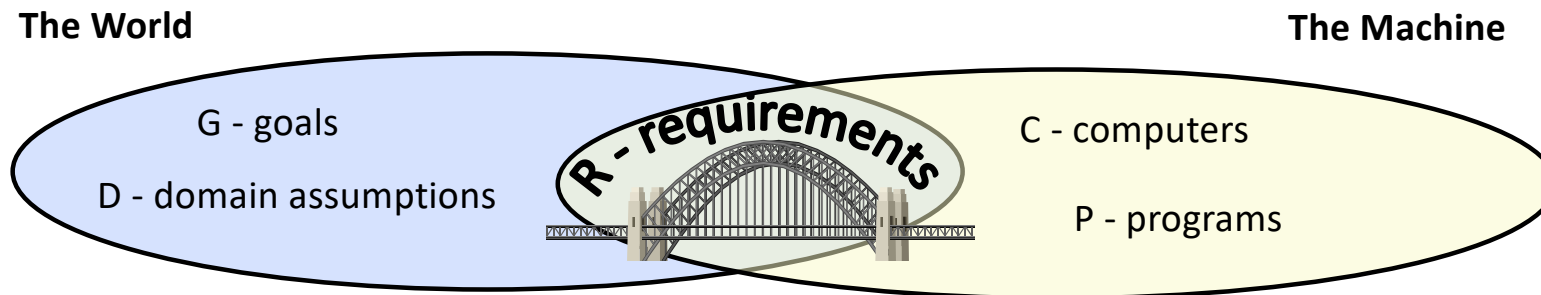
- Constraints on **how functionality must be provided** to the end user
- Application domain determines
 - Their **relevance**
 - Their **prioritization**
- **Examples:**
 - Relevant nonfunctional requirements for Netflix?
 - Relevant nonfunctional requirements for Ariane 5?
- Have a **strong impact** on the **structure** of the system to be
 - Example: a system requires 24/7 availability → it is likely to be thought as a replicated system (with redundant components)

Exercise on recognizing requirements: are these requirements?

- The user should insert correct information in the enrolment form
- The system should check whether fiscal code are well formed

Exercise on recognizing requirements: are these requirements?

- The user should insert correct information in the enrolment form
 - How can the software prevent a user to insert incorrect information? -> It is a domain assumption!
- The system should check whether fiscal code are well formed
 - Yes, the software can do this -> It is a requirement!





Examples of bad requirements

- “The system shall validate and accept credit cards and cashier’s check with high priority”
- **Issue(s)?**
 - **Multiple concerns**: two requirements instead of one
 - **Ambiguous**: if the credit card processing works, but the cashier’s check validation does not, is this requirement satisfied?
 - Should not, but this is misleading...
 - **Ambiguous**: does “high priority” refer to cashier’s checks only, so are credit cards low priority? Other way around? Are they both high priority?



Examples of bad requirements

- “The system shall process all mouse clicks very fast to ensure users do not have to wait”
- **Issue(s)?**
 - **Cannot be verified (tested):** what does “fast” mean? Do we have a metric? Can you quantify it?

Examples of bad requirements

- “the user must have Adobe Acrobat installed”
- **Issue(s)?**
 - **Cannot be achieved** by the software system itself: it is not something the system must do
 - It could be expressed as domain assumption, but it is not a functional requirement for our software

Eliciting requirements

Elicitation and modeling: how to cope with this complex task?

- Adopting various approaches and strategies (listening, observing, studying, ...) and combining the results achieved with all of them
- Being as close as possible to stakeholders
- Letting stakeholders describing their viewpoints



A simple and effective tool: 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 to be



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 appears 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 time of arrival (ETA) to Alice.*
- *Alice received the acknowledgment and the ETA.*

Heuristics for finding scenarios

- Ask yourself or the client the following questions:
 - Which user groups are supported by the system to perform their work?
 - 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 possible
 - Ask to speak to the end user, not just to the software contractor
 - Expect resistance and try to overcome it



From scenarios to use cases

- Scenarios provide a nice summary of what the requirements analysis team can derive from
 - Observation
 - Interviews
 - Analysis of documentation
- ... they can be very specific
- How to abstract from details and specificities?
 - ... **Use cases!**

Use cases

- Generalize scenarios
 - Example: from “Warehouse on fire” we can generalize a “Report Emergency” use case
- Structure the description in terms of
 - Participating actors
 - Describe the Entry Condition
 - Describe the Flow of Events
 - Describe the Exit Condition
 - Describe Exceptions
 - Describe Special Requirements (Constraints, Nonfunctional Requirements)

Use case example: ReportEmergency

- Participating actors:
 - Field Officer (Bob and Alice in the Scenario)
 - Dispatcher (John in the Scenario)
- Entry Condition: True (an emergency can be reported all times)

Use cases

- Generalize scenarios
 - Example: from “Warehouse on fire” we can generalize a “Report Emergency” use case
- Structure the description in terms of
 - Participating actors
 - Describe the Entry Condition
 - Describe the Flow of Events
 - Describe the Exit Condition
 - Describe Exceptions
 - Describe Special Requirements (Constraints, Nonfunctional Requirements)

Use case example: ReportEmergency

- Flow of events
 - The **FieldOfficer** activates the “Report Emergency” function of her terminal.
 - **FRIEND** (the system to be developed) 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 allocates resources by invoking the AllocateResources use case. The Dispatcher selects a response and acknowledges the emergency report.

Use cases

- Generalize scenarios
 - Example: from “Warehouse on fire” we can generalize a “Report Emergency” use case
- Structure the description in terms of
 - Participating actors
 - Describe the Entry Condition
 - Describe the Flow of Events
 - Describe the Exit Condition
 - Describe Exceptions
 - Describe Special Requirements (Constraints, Nonfunctional Requirements)



Use case example: ReportEmergency

- Participating actors: Field Officer, Dispatcher
- Entry Condition: True
- Flow of events:
- Exit condition: The FieldOfficer has received the acknowledgment and the selected response.

Use cases

- Generalize scenarios
 - Example: from “Warehouse on fire” we can generalize a “Report Emergency” use case
- Structure the description in terms of
 - Participating actors
 - Describe the Entry Condition
 - Describe the Flow of Events
 - Describe the Exit Condition
 - Describe Exceptions
 - Describe Special Requirements (Constraints, Nonfunctional Requirements)



Use case example: ReportEmergency

- Exceptions:
 - The FieldOfficer is notified immediately if the connection between her terminal and the control room is lost
 - The Dispatcher is notified immediately if the connection between any logged in FieldOfficer and the control room is lost
- 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



Example of a poor use case

- Use case name: Accident
 - Participating Actors:
 - Field Officer
 - Flow of Events:
 - 1. The Field Officer reports the accident
 - 2. An ambulance is dispatched
 - 3. The Dispatcher is notified when the ambulance arrives on site
- (not an action, i.e., a verb)
- (Dispatcher actor is missing here but mentioned in the next section)
- (by whom?)
- (Who notifies the Dispatcher?)

Are we done at this point?



Tips to define proper use cases

- Use cases named with verbs that indicate what the user is trying to accomplish
- Actors named with nouns
- Use cases steps in active voice
- The causal relationship between steps should be clear
- A use case per user transaction
- Separate description of exceptions
- Keep use cases small (no more than two/three pages)
- The steps accomplished by actors and those accomplished by the system should be clearly distinguished
 - This helps us in identifying the boundaries of the system

Use case example: AllocateResources

- **Actors:**
 - **Dispatcher:** the Dispatcher allocates a resource to an Emergency if the resource is available (of course, he also updates and removes Emergency Incidents, Actions, and Requests in the system)
 - **Resource Allocator:** The Resource Allocator is responsible for allocating resources in case they are scarce
 - **Resources:** The Resources that are allocated to the Emergency

Allocate a resource (1)

- Use case name: AllocateResources
- Participating Actors:
 - Dispatcher (John in the Scenario)
 - Resource Allocator
 - Resources
- Entry Condition
 - An Incident has been opened
- Flow of Events
 - The Dispatcher selects the types and number of Resources that are needed for the incident
 - FRIEND replies with a list of Resources that fulfill the Dispatcher's request
 - The Dispatcher selects the Resources from the list and allocates them for the incident
 - FRIEND automatically notifies the Resources
 - The Resources send a confirmation

Allocate a resource (2)


- Exit Condition
 - The use case terminates when the resource is committed.
 - The selected Resource is now unavailable to any other Emergences or Resource Requests
- Exceptions
 - If the list of Resources provided by FRIEND is insufficient to fulfill the needs of the emergency, the Dispatcher informs the Resource Allocator
 - The Resource Allocator analyzes the situation and selects new Resources by decommitting them from their previous work
 - FRIEND automatically notifies the Resources and the Dispatcher
 - The Resources send a confirmation

How to specify a use case (summary)

- Name of Use Case
- Actors
 - Description of Actors involved in use case
- Entry condition
 - “When this use case starts the following condition is true...”
- Flow of Events
 - Free form, informal natural language
- Exit condition
 - “This use case terminates when the following condition holds...”
- Exceptions
 - Describe what happens if things go wrong
- Special Requirements
 - Nonfunctional Requirements, Constraints

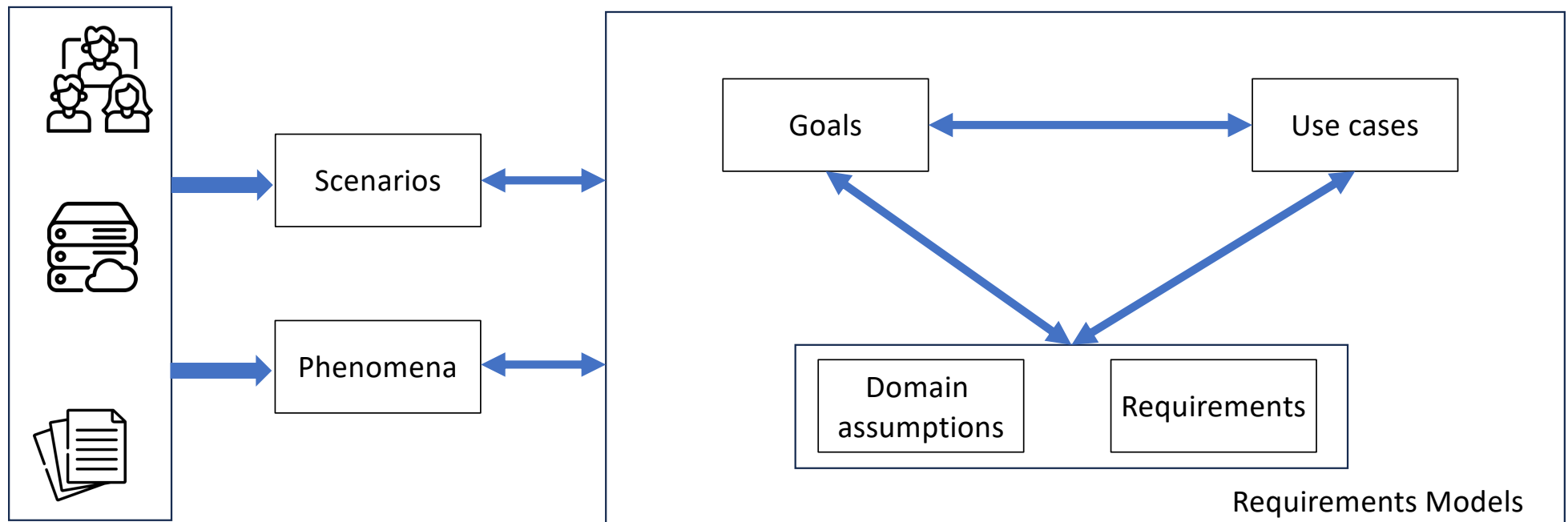
From use cases to requirements and assumptions

- Each use case may lead to one or more requirements
- Examples from ReportEmergency and AllocateResources
 - FRIEND shall support Field Officers in reporting an emergency
 - FRIEND must show a response time lower than 30 seconds when reacting to emergency-related requests
 - FRIEND shall support Dispatchers in allocating the resources to the incident
 - FRIEND shall allow the dispatcher to call the ResourceAllocator if needed
 - The ResourceAllocator has the authority to make decisions on resource decommission



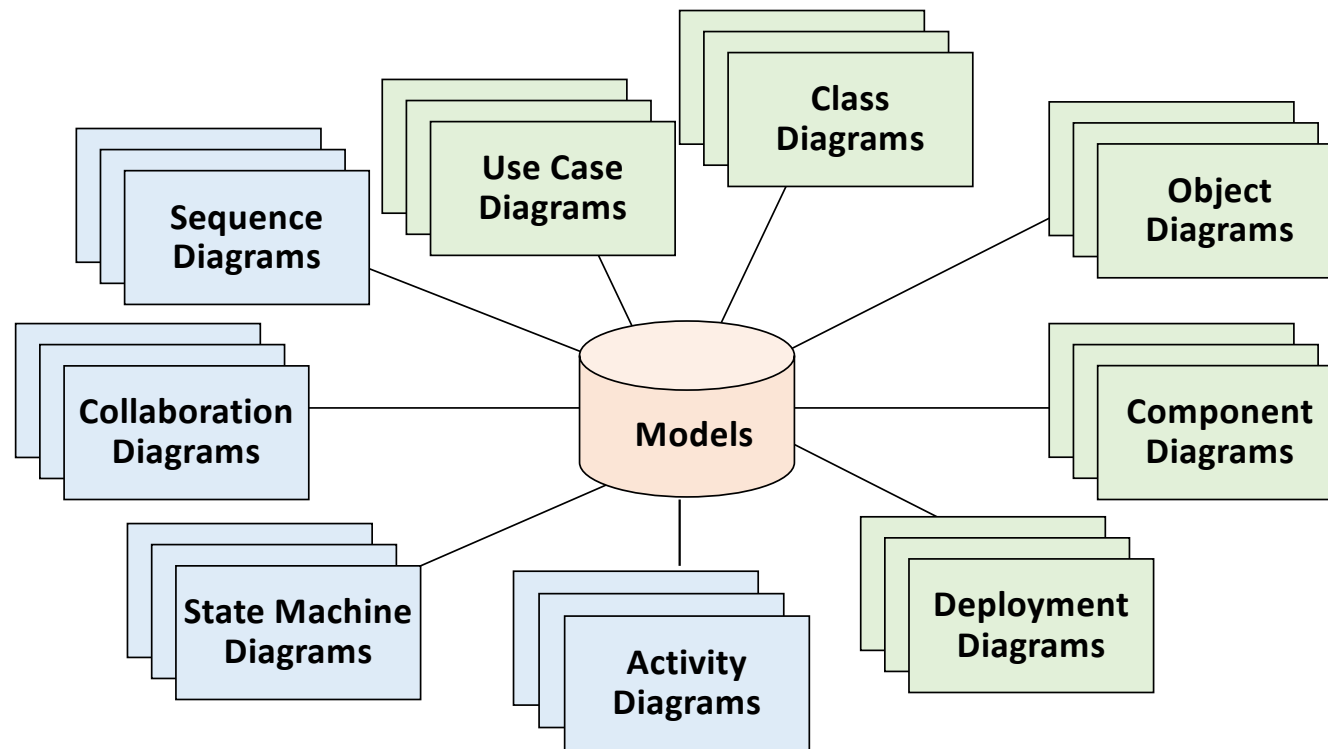
Guess the meaning
of the color
code 😊

Relating all ingredients together

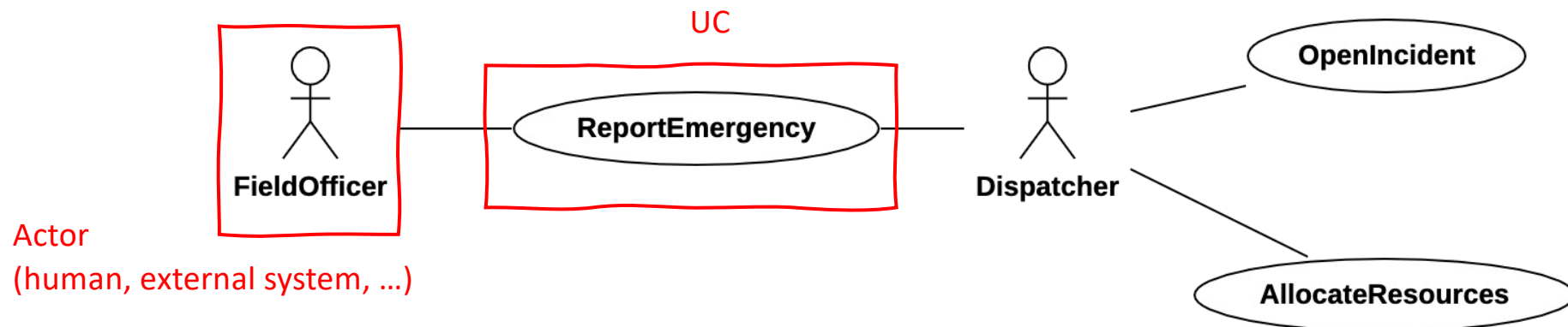


The role of UML in requirement modeling

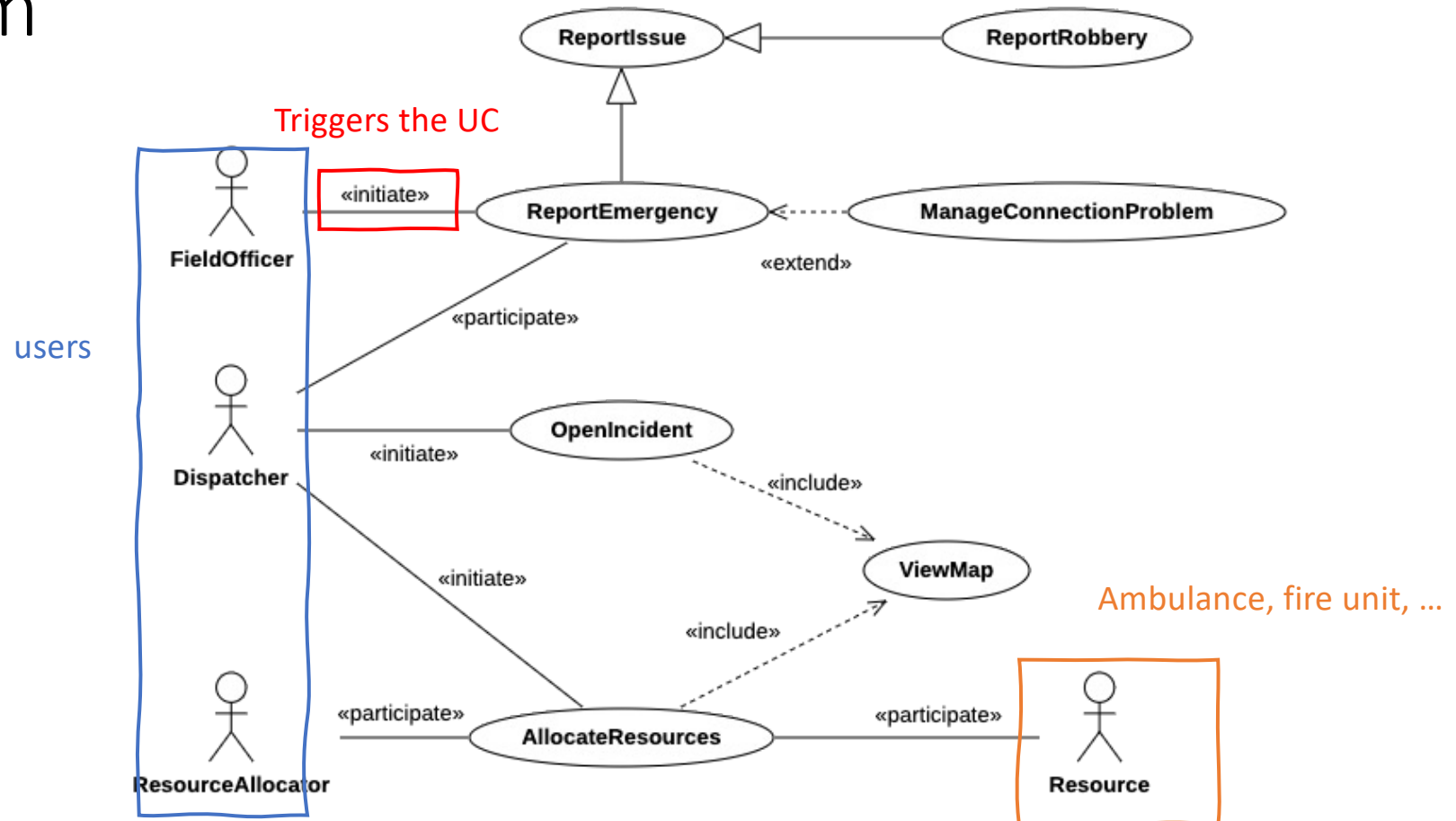
UML Models, Views, and Diagrams



UML Use Case Diagrams: Incident Management System Example



Example: refined Incident Management System





Use case relationships compared

Generalization	Extend	Include
Base UC Specialized UC	Extending UC 	Included UC
Base use case could be abstract use case (incomplete) or concrete (complete).	Base use case is complete (concrete) by itself, defined independently.	Base use case is incomplete (abstract use case).
Specialized use case is required, not optional, if base use case is abstract.	Extending use case is optional, supplementary.	Included use case required, not optional.
No explicit location to use specialization.	Has at least one explicit extension location.	No explicit inclusion location but is included at some location.
No explicit condition to use specialization.	Could have optional extension condition.	No explicit inclusion condition.

Extend: <https://www.uml-diagrams.org/use-case-extend.html>

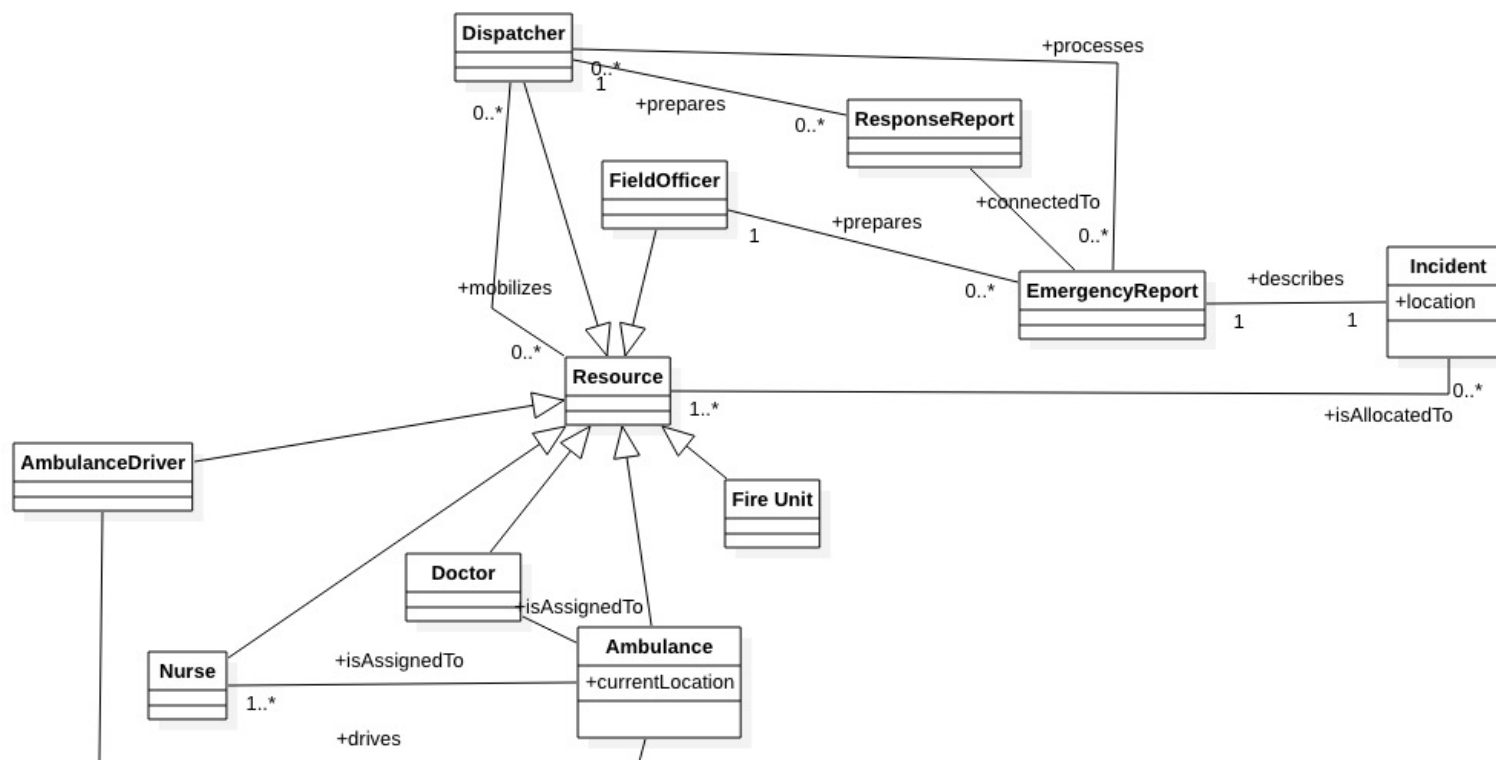
Include: <https://www.uml-diagrams.org/use-case-include.html>



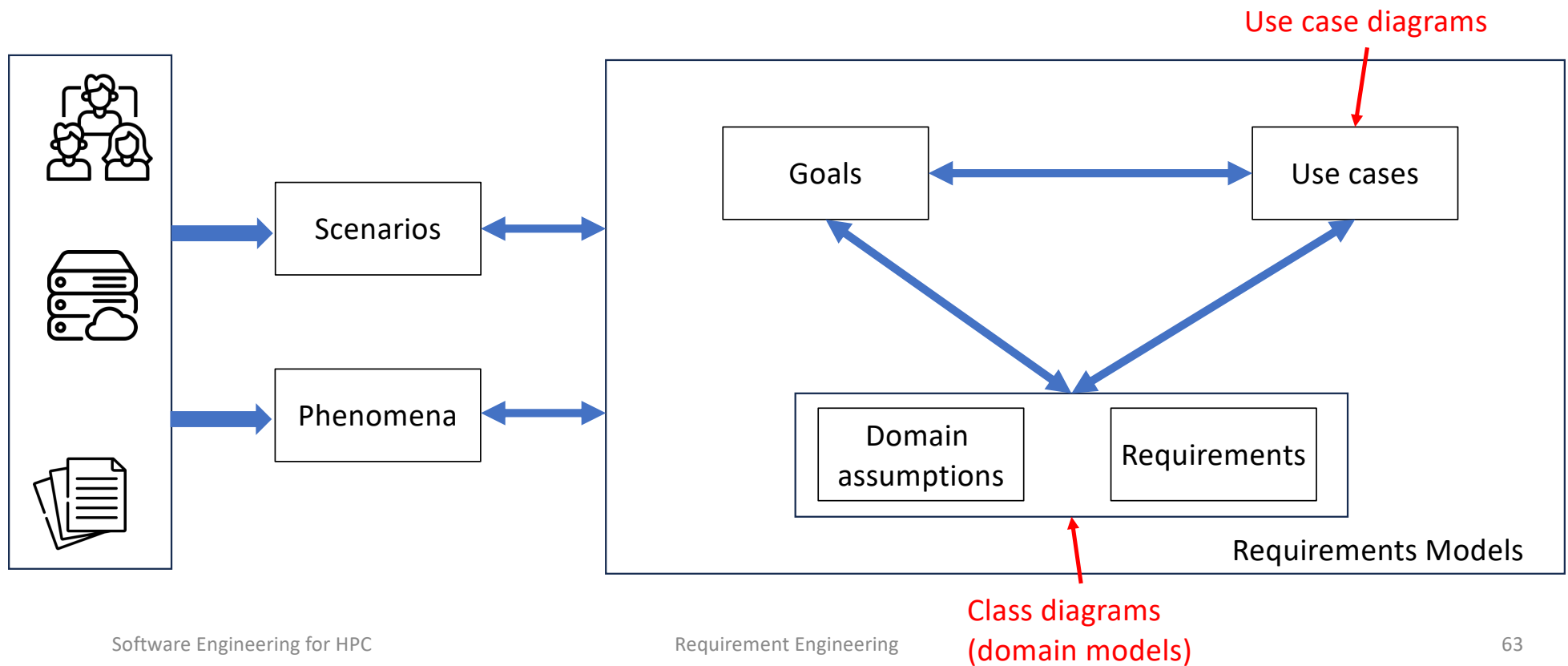
Requirements-level class diagrams

- They are **conceptual models** for the application **domain**
- Include the entities relevant to the application domain, their attributes and the relationships among them

Example of class diagram from the Incident Management System



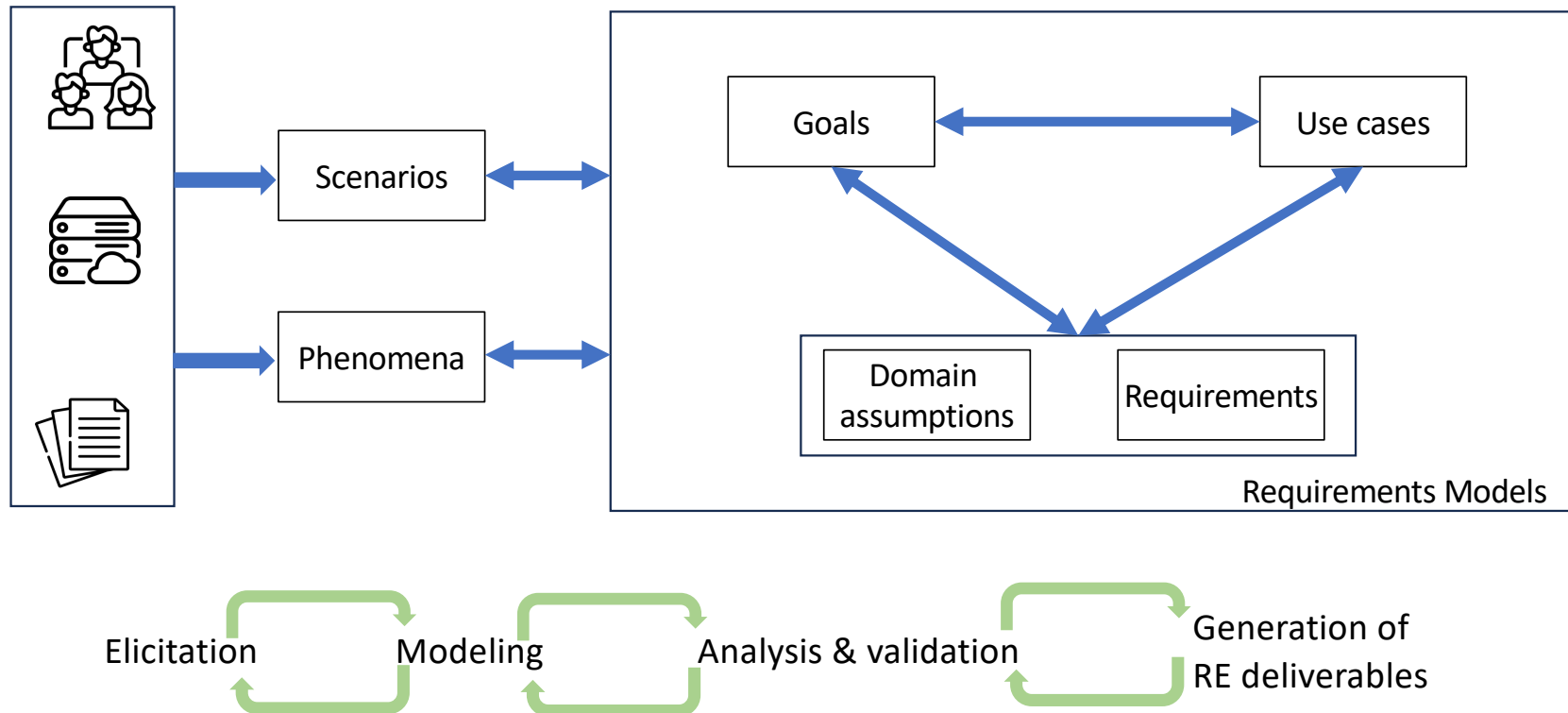
Relating all ingredients together



The requirement engineering process and the RASD



The requirement engineering process



Possible Requirement Analysis and Specification Document (RASD) structure

1 Introduction

- Purpose
- Scope
- Definitions, acronyms, abbreviations
- Reference documents
- Overview

2 Overall Description

- Scenarios
- User characteristics
- Analysis of phenomena
- Use cases
- Requirements
- Domain Assumptions

3 Bibliography

Adapted from ISO/IEC/IEEE 29148 dated Dec 2011

Summary

- Requirement engineering has a crucial role in the software life cycle
 - Ultimate goal: understand what the software to be should do
 - A possible approach:
 - Think at scenarios,
 - Define use cases,
 - From these derive entities, phenomena, requirements, domain assumptions
 - Write the RASD
 - Share and discuss it with the other stakeholders
 - Modify and reiterate it