POLITECNICO
MILANO 1863

Software Engineering 2

Introduction to Requirements Engineering (RE)

The world and the machine

The Airbus incident

Requirements Engineering (RE)

Context, Definitions, Importance and difficulties, RE process

Context: where do we find RE?

Feasibility Study

Requirements Analysis & Specification

Design

Coding & Unit Test

Integration & System Test

Deployment

Maintenance

- Not to be forgotten in all other phases, too!
- **Example**: as the system is deployed, new requirements emerge!

Requirements Engineering: definition
[Nuseibeh&Easterbrook '00]

POLITECNICO MILANO 1863

- 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 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

What is a requirement?

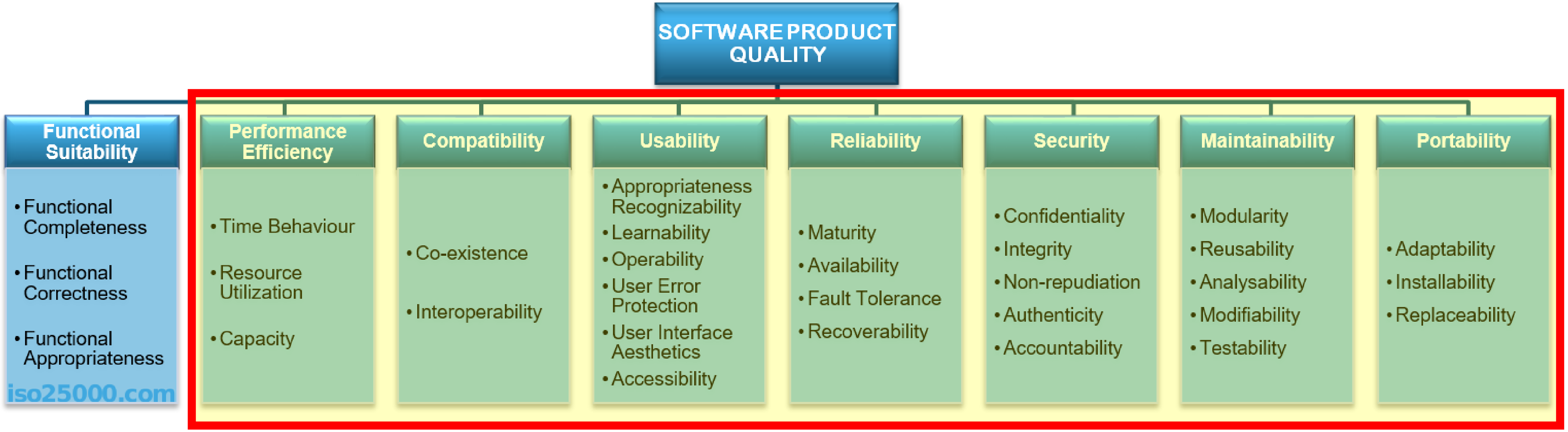
- Examples of candidate requirements
  - “The system shall allow users to reserve taxis”
  - “The system has to provide a feedback in 5 seconds”
  - “The system should never allow non-registered users to see the list of other users willing to share a taxi”
  - “The system should be available 24/7”
  - “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”

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)

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 a constraint or assumption, but it is not a functional requirement

Exercise

- Consider the following 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"
- Classify them by their kind (functional, nonfunctional and constraints)
- Highlight bad requirements

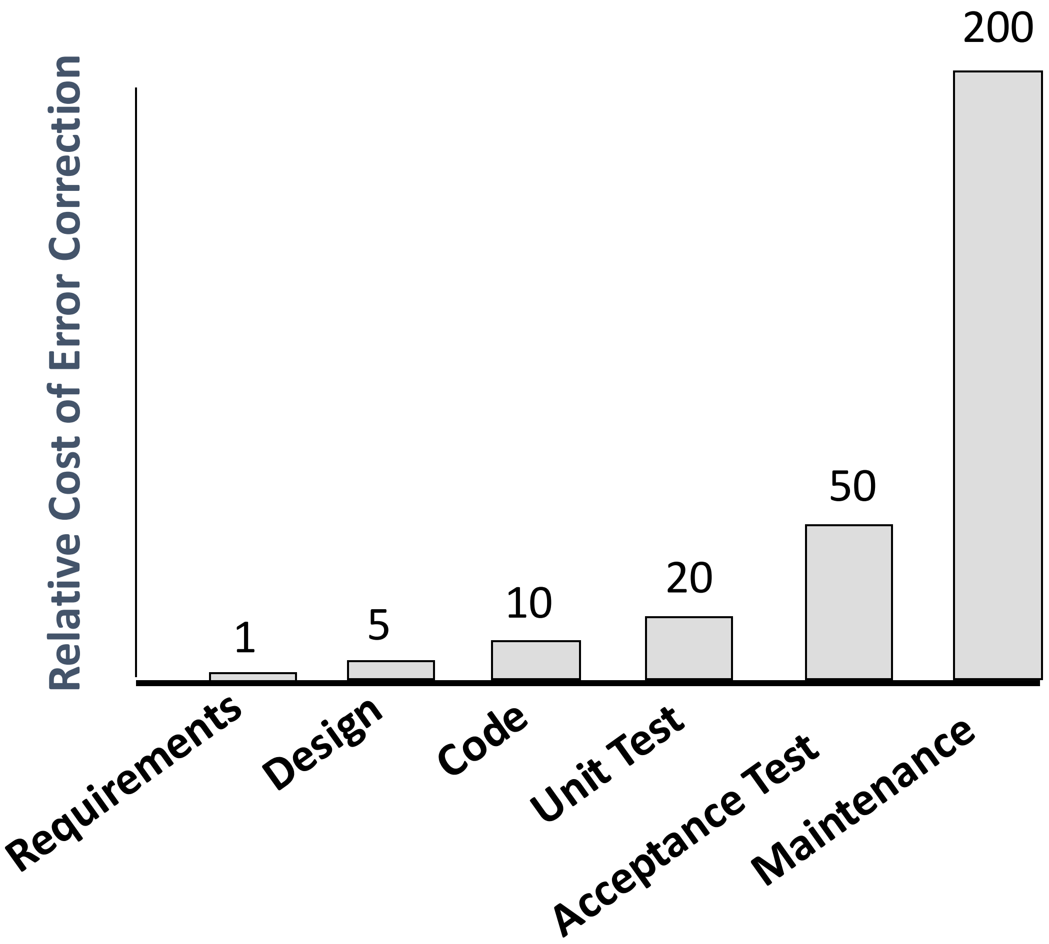
Issues concerning RE

- Poor requirements are ubiquitous ...
  - “The system should guarantee that the reserved taxi picks the user up”: is this reasonable?
  - “[…] requirements need to be engineered and have continuing review & revision” (Bell & Thayer, empirical study, 1976)
- RE is hard & critical ….
  - “[…] hardest, most important function of SE is the iterative extraction & refinement of requirements” (F. Brooks, 1987)

Cost of late correction (Boehm, 1981)

POLITECNICO MILANO 1863

- The cost of correcting an error depends on the number of subsequent decisions that are based on it
- Errors in requirements have the potential for greatest cost, because many other design decisions depend on them



What makes RE so complex ?

- Broad scope
  - **Composite systems**
    - human organizations + physical devices + software components
  - **More than one system**
    - alternative proposals for system-to-be, system evolutions, product family
  - **Multiple abstraction levels**
    - high-level goals, operational details
- Multiple concerns
  - functional, quality, development
  - hard and soft concerns

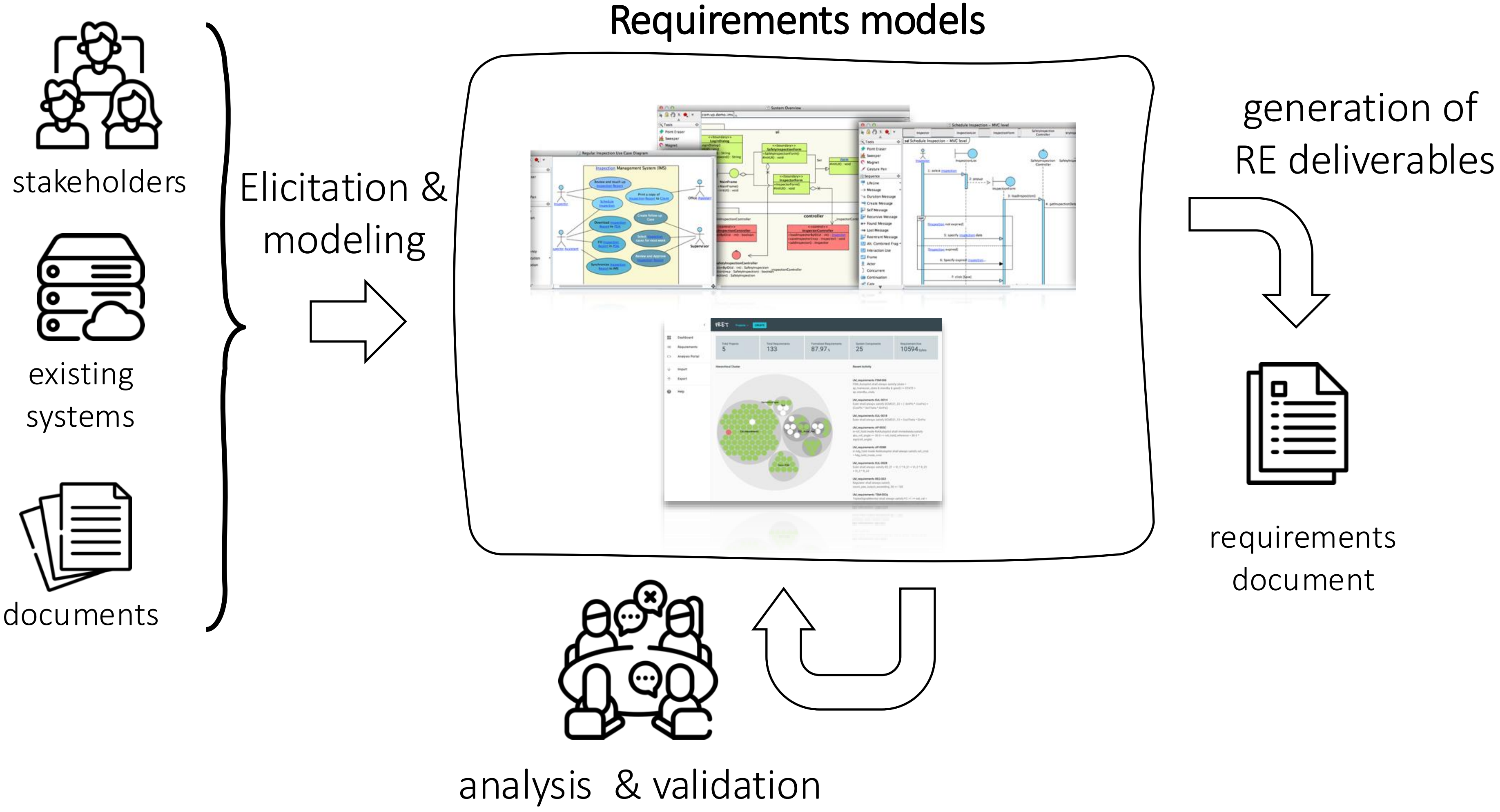
$\Longrightarrow$ conflicts

- Multiple stakeholders with different background
  - customers, users, domain experts, developers, ...

$\Longrightarrow$ conflicts

RE workflow

POLITECNICO MILANO 1863



Requirements Engineering (RE)

Understanding phenomena and requirements: the World and the Machine

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?
  - What are the "correctly encoded" 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?

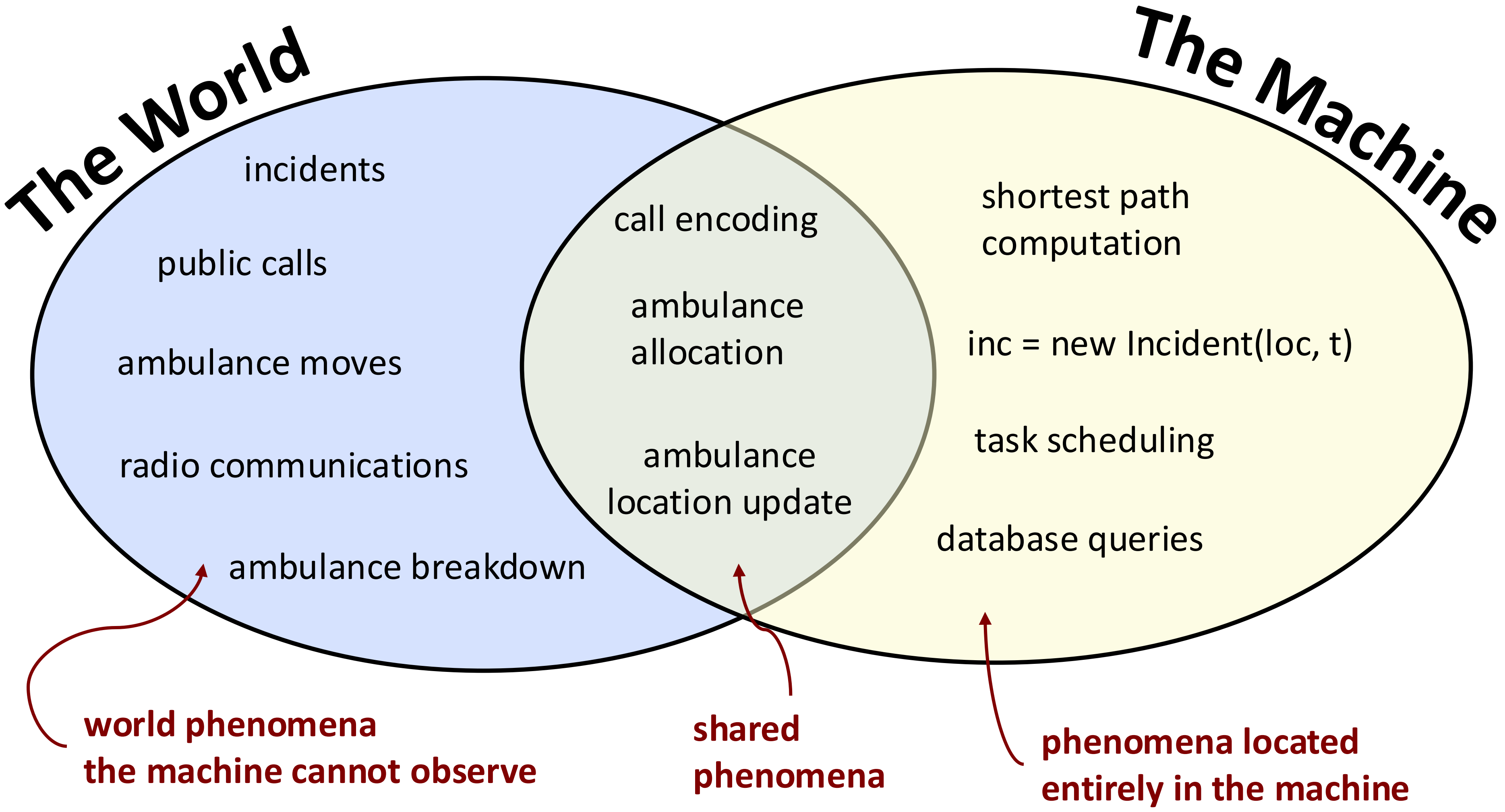
The World and the Machine
(M. Jackson & P. Zave, 1995)

- Terminology
  - **Machine** = the portion of system to be developed
    - typically, software-to-be + hardware
  - **World** (or environment) = the portion of the real-world affected by the machine
- The purpose of the machine is always in the world
  - Examples
    - Ambulance Dispatching System
    - Banking Application
    - Word Processor
    - …

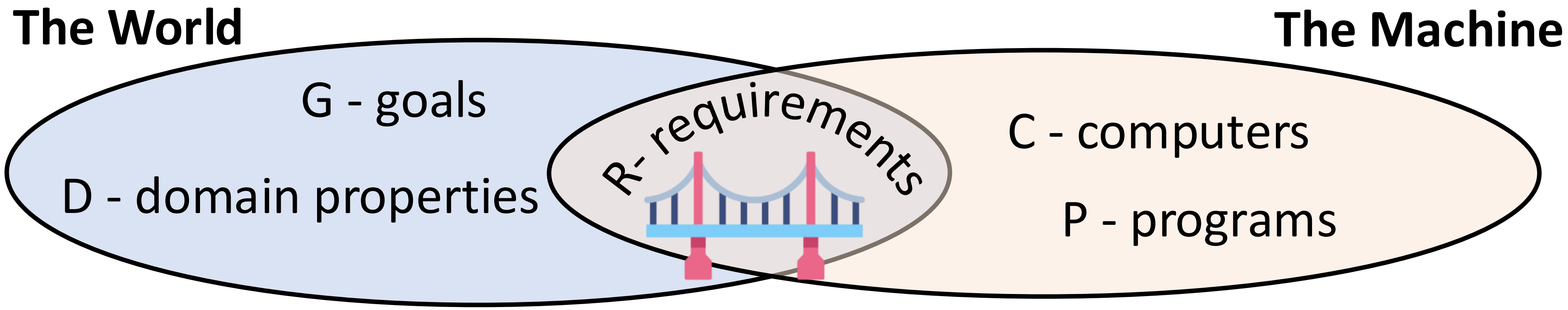
World and machine phenomena

- 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, requirements: an example

- **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 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 an existing mobile data terminal”
- **Requirement**
  - “When a call reporting a new incident is encoded, the Automated Dispatching Software should send the nearest available ambulance according to information available from location updates and availability notifications”

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**

    $$\mathbf{R \text{ and } D \models G}$$

    - 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

Requirements Engineering (RE)

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

The A320 example

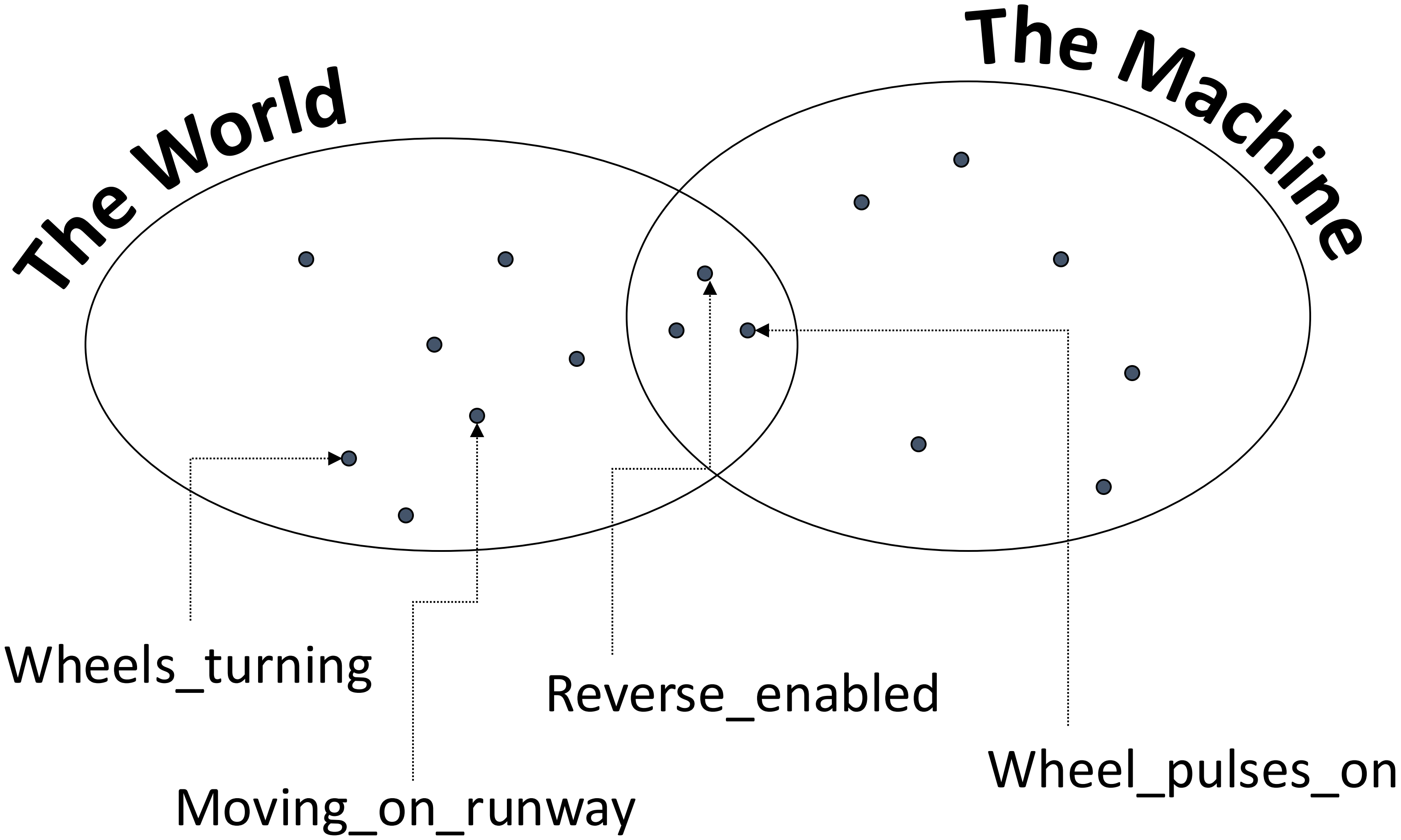
Example – Airbus A320

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

- [Sep 14, 1993] A Lufthansa Airbus on a flight from Frankfurt to Warsaw landed in bad weather conditions (rain and wind)
- On landing, the aircraft's software-controlled braking system deployed with a delay of 9 seconds after touch down
- There was insufficient runway remaining to stop the plane and the aircraft ran into a grass embankment
- **Result**: 2 people were killed, 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

Example – Airbus A320 Braking Logic



Goal, domain assumptions, requirement

- **Goal G**
  - “Reverse thrust enabled iff the aircraft is moving on the runway”
- **Domain Assumptions D**
  - “Wheel pulses on iff wheels turning”
  - “Wheels turning iff moving on runway”
- **Requirement R**
  - “Reverse thrust enabled iff wheel pulses on”

- **Verification**: R and D $\models$ G holds

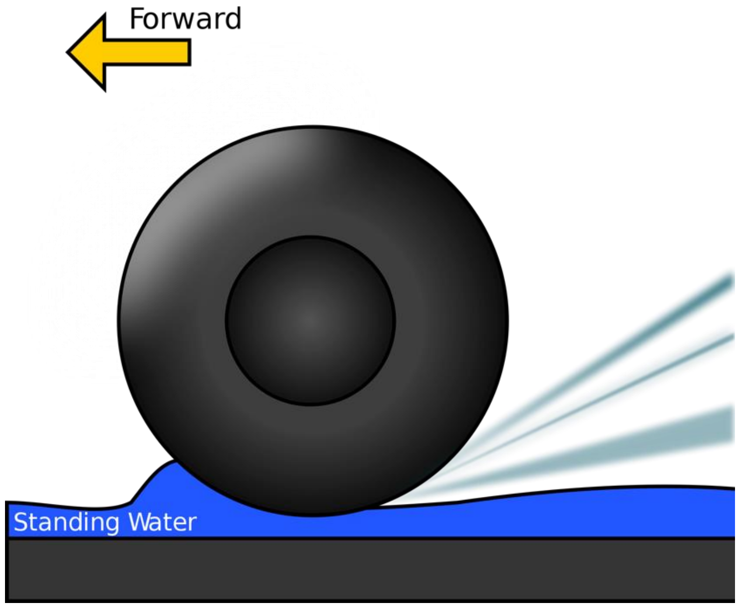
Correctness argument

- Goal
  - Reverse_enabled ⇔ Moving_on_runway
- Domain properties
  - Wheel_pulses_on ⇔ Wheels_turning
  - Wheels_turning ⇔ Moving_on_runway
- Requirement
  - Reverse_enabled ⇔ Wheels_pulses_on

- We can prove that $R \land D \models G$ → it seems the system is correct by design
  - What happened?
  - D are not valid assumptions!
  - **Invalid domain assumptions → Warsaw accident**

Problem: Aquaplaning

- Domain assumptions D do not hold in reality, the correctness argument is flawed!
- Incorrect domain assumptions lead to disasters!



Requirements Engineering (RE)

Example of goals, assumptions, requirements

The Turnstile Control System (TCS)

- Main purpose of the system: let people in only if they pay (with a coin for simplicity)

Goals

G1: At any time entries should never exceed cumulated payments (for simplicity, assume 1 coin for 1 entrance)

G2: Those who pay are not prevented from entering (by the “machine”)

- They are **optative** descriptions
- Both are said to be safety properties
  - They state that nothing bad will ever occur

Next steps

1. Carry out domain analysis
   - Identify phenomena, in particular the shared ones
   - Identify relationships among phenomena
     - Useful to identify domain assumptions
   - Identify domain assumptions
2. Define requirements
   - … with an eye towards the goals to be achieved
   - Make sure that requirements are expressed using only shared phenomena
3. Prove that R and D $\models$ G
   - i.e., requirements + domain assumptions guarantee goal satisfaction

Identify relevant phenomena

POLITECNICO
MILANO 1863

- "Relevant" with respect to the purposes of the system
  - i.e., control people entrance

world phenomena
shared phenomena
environment controlled
machine controlled

Coin: coin inserted
Push: person pushes turnstile
Turn: turnstile turns
Enter: person enters the room
Lock:
Unlock:
} electric signals

Relationships among phenomena

- Events Push, Turn and Enter are clearly linked with each other
- These links are outside of the control of the machine (our control software TCS)
- …and are captured by Domain Assumptions
  - D_a: Turn occurs only after a Push occurs
  - D_b: Turn leads to Enter
  - D_c: Enter occurs only after Turn occurs
- That is, each Enter corresponds to a Turn; on the other hand, there can be Pushes that do not lead to Turns
- To study TCS, no need to deeply study when a person pushes the turnstile, but it does not turn
- We conflate the Push and Turn events in a single Push&Turn **shared, world-controlled** event
  - It is like adding a new assumption "D_d: Push leads to Turn"

Domain assumptions

D1a: Enter cannot occur without Push&Turn

D1b: a new Push&Turn cannot occur until the previous visitor entered

- Can be summed up in the following assumption
  - **D1**: Push&Turn and Enter alternate, starting with Push&Turn

D2: Push&Turn always leads to Enter

- enforced by a hydraulics system

D3: Push&Turn cannot occur if, and only if, the turnstile is Locked

D4: After Push&Turn, there is a minimum delay *d* before the next Push&Turn can occur

Let us revisit goal G1

G1: At any time entries (e) should never exceed accumulated payments (c)

- i.e., **$e \leq c$ must hold**

- **G1** can be enforced by controlling either entries or coins
  - the machine cannot force Coin events
  - it can prevent Enter events (through Lock)

Requirements for G1

G1: **e ≤ c must hold**

R1: TCS must give the unlock command only if the number of Push&Turn events (p) is less than the number of accumulated payments (c)

- i.e., only if **p < c holds**

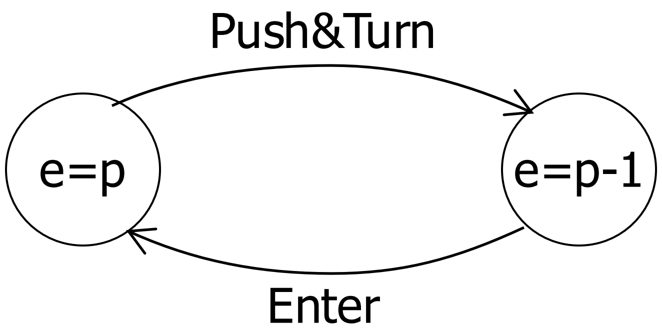
R2: If Push&Turn occurs and p = c holds, TCS must give command Lock within *d* time

- i.e., before the next Push&Turn can occur

R3: TCS must lock the turnstile at startup time

Show that G1 holds (1)

- We show that R1, R2, R3, D1, D3, D4 guarantee G1
  - R1, R2, R3, D1, D3, D4 ⊨ G1
- Notice that, from D1, we have:



e ≤ p holds

e number of Enter events

p number of Push&Turn events

- We need to show that e ≤ c holds, so if we can show that p ≤ c holds, we have that e ≤ p ≤ c, which is the desired property

Show that G1 holds (2)

- We show that Push&Turn can occur only if $p < c$ holds:
  - Initially $p = c = 0$, and the turnstile is locked (from R3), so Push&Turn cannot occur (from D3)
  - Turnstile is unlocked only if $p < c$ holds (from R1), so the first Coin must occur before the first Push&Turn can occur
  - Consider now a time in which $p < c$ holds, and Push&Turn occurs (we indicate with p' the new number of pushes, so $p' = p+1$)
    - If $p' = c$, then (from D4) some time *d* must pass before a new Push&Turn can occur and (from R2) the Lock command is given before time interval *d* passes
    - Hence, a new Push&Turn cannot occur (from D3) , unless a new coin is inserted (i.e., unless the number of Push&Turn is again less than the number of Coin)

Let us consider goal G2

G2: Those who pay are not prevented from entering (by the "machine")

- That is, we need to show that, if **p < c holds, then Enter can occur**

- Intuitively, we should not lock the turnstile if there is an excess of payments, as captured by the following new requirements

R4: TCS must lock the turnstile only if p = c holds

R5: If p < c holds and the turnstile is locked, TCS must unlock the turnstile

Show that G2 holds

- We show that R4, R5, D1, D2, D3 guarantee G2
  - R4, R5, D1, D2, D3 $\models$ G2
- We show that if $p < c$ holds then Enter can occur:
  - From D1, D2, we have that Enter occurs if, and only if, Push&Turn occurs
  - Then, from D3 we have that Enter can occur if, and only if, the turnstile is unlocked
  - So, to prove G2 we need to show that if $p < c$ holds, then the turnstile is unlocked
  - Consider now a time in which $p < c$ holds, we have 2 cases
    - If the turnstile is unlocked, then from R4 we have that the machine does not lock it, so it stays unlocked
    - If the turnstile is locked, then from R5 we have that the machine unlocks it

Assignments for the next week classes

- Watch the videos you find here
  - On requirement elicitation: https://polimi365-my.sharepoint.com/:v:/g/personal/10143828_polimi_it/EQzGESnSzdBMnTIQvI9XG6QBV0zp715j6HqGqhHJ2Vx7qQ?e=Rx4Cab
  - On modeling: https://polimi365-my.sharepoint.com/:v:/g/personal/10143828_polimi_it/EfuNIeNgPwtIrwcJQNI1s2YBw6G0vEhQDw4yNbpqLH4wQg?e=per3xm
- They will be used as a basis for discussion in class next time(s)

Summary

- The boundary between the World & the Machine is generally not given at the beginning of the development process
- The purpose of a RE activity is
  - to identify the real goals of the project
  - to explore alternative ways to satisfy the goals, through:
    - alternative interfaces between the world and the machine
    - alternative pairs (Req, Dom) such that
      Req and Dom $\models$ G
  - to evaluate the strengths and risks of each alternative, in order to select the most appropriate one

References

- B. Paech, "What Is a Requirements Engineer?" in IEEE Software, vol. 25, no. 04, pp. 16-17, 2008.
- "ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering," in ISO/IEC/IEEE 29148:2018(E) , vol., no., pp.1-104, 30 Nov. 2018, doi: 10.1109/IEEESTD.2018.8559686.
- Bashar Nuseibeh and Steve Easterbrook. 2000. "Requirements engineering: a roadmap". In Proceedings of the Conference on The Future of Software Engineering (ICSE '00). Association for Computing Machinery, New York, NY, USA, 35–46. DOI:https://doi.org/10.1145/336512.336523
- P. Zave, "Classification of research efforts in requirements engineering," Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95), 1995, pp. 214-216, doi: 10.1109/ISRE.1995.512563.
- Michael Jackson. 1995. "The world and the machine". In Proceedings of the 17th international conference on Software engineering (ICSE '95). Association for Computing Machinery, New York, NY, USA, 283–292. DOI:https://doi.org/10.1145/225014.225041
- M. Jackson and P. Zave, "Deriving Specifications from Requirements: an Example," 1995 17th International Conference on Software Engineering, 1995, pp. 15-15, doi: 10.1145/225014.225016.