POLITECNICO
MILANO 1863

Requirements Engineering (RE)

Context

Definitions

Importance and difficulties

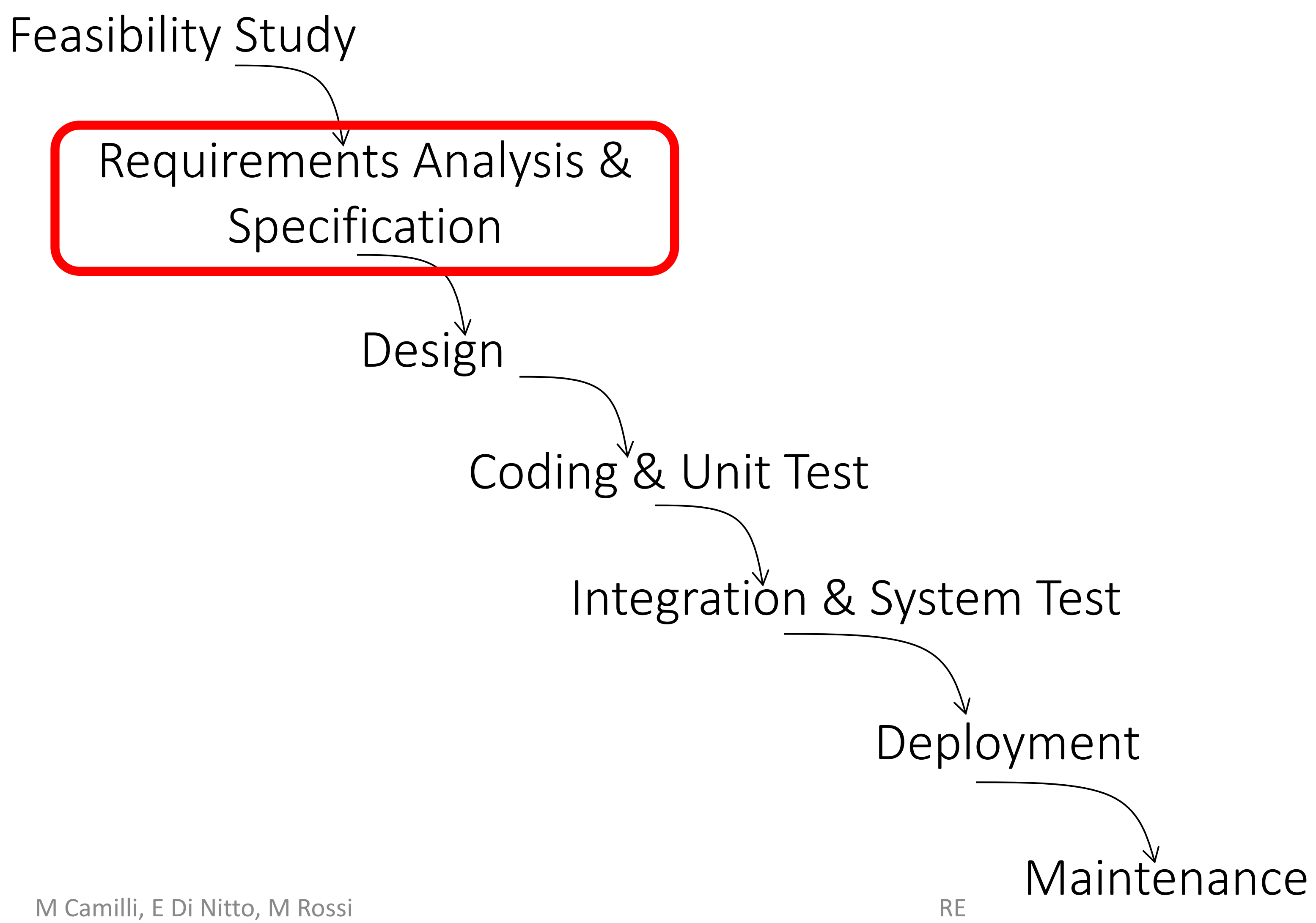
The RE process

Understanding the world-machine relationships

The Airbus incident

M Camilli, E Di Nitto, M Rossi

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!
- As the system is deployed, new requirements emerge!

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

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”

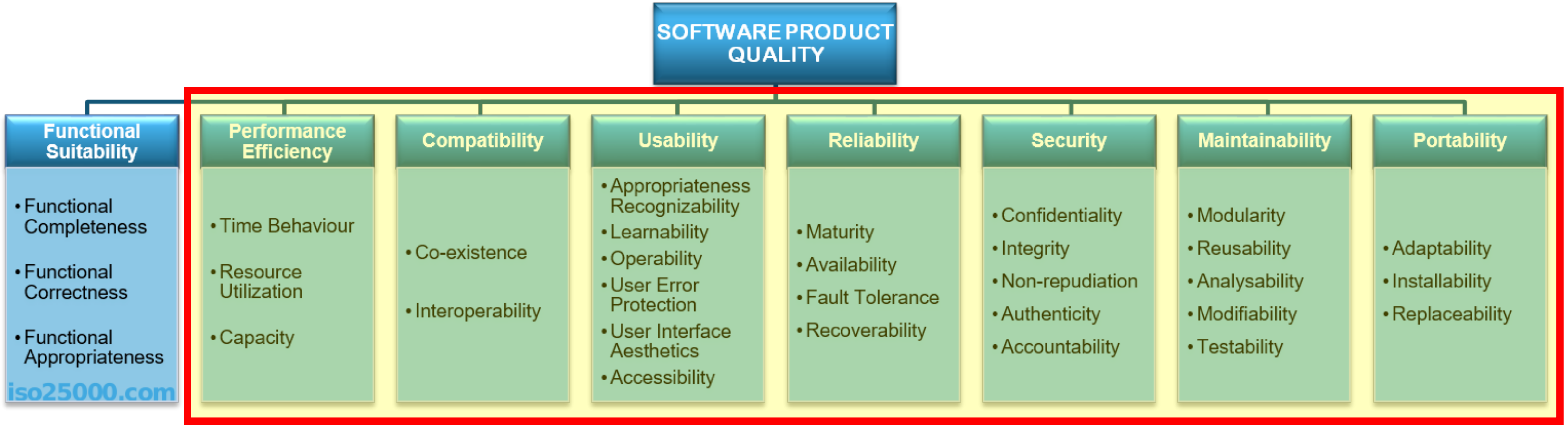
Types of requirements

- **Functional** requirements:
  - Describe the interactions between the system and its environment independent from implementation
  - Examples:
    - "A word processor user should be able to search for strings in the text"
    - "The system shall allow users to reserve taxis"
  - Are the main goals the software to be has to fulfill
- **Non-functional** requirements:
  - User-visible aspects of the system not directly related to functional behavior
  - Examples:
    - "The response time must be less than 1 second"
    - "The 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"
    - "The credit card payment system must be able to be dynamically invoked by other systems relying on it"

Characteristics of non-functional requirements

- Constraints on how functionality has to be provided to the end user
- Independent of the application domain
- … but the application domain determines
  - Their relevance
  - Their prioritization
- Have a strong impact on the structure of the system to be
  - Example: if a system has to guarantee to be available 24 hours per day, it is likely to be thought as a (at least partially) replicated system

Non-functional requirements and product qualities



Examples of bad requirements

- "The system shall validate and accept credit cards and cashier’s checks… high priority"
- "The system shall process all mouse clicks very fast to ensure users do not have to wait"
- "The user must have Adobe Acrobat installed"

Examples of bad requirements

- "The system shall validate and accept credit cards and cashier’s checks… high priority"
  - **Problem**: two requirements instead of one
    - If the credit card processing works, but the cashier’s check validation does not, is this requirement satisfied or not?
      - Has to be not satisfied, but that is misleading
    - Maybe only credit cards are high priority and cashier’s checks are low priority
- "The system shall process all mouse clicks very fast to ensure users do not have to wait"
  - **Problem**: this is not testable… quantify how fast is acceptable?
- "The user must have Adobe Acrobat installed"
  - **Problem**: this is not something our system must do
  - It could be in the constraints/assumptions or maybe operating environment sections, but it is not a functional requirement of our system

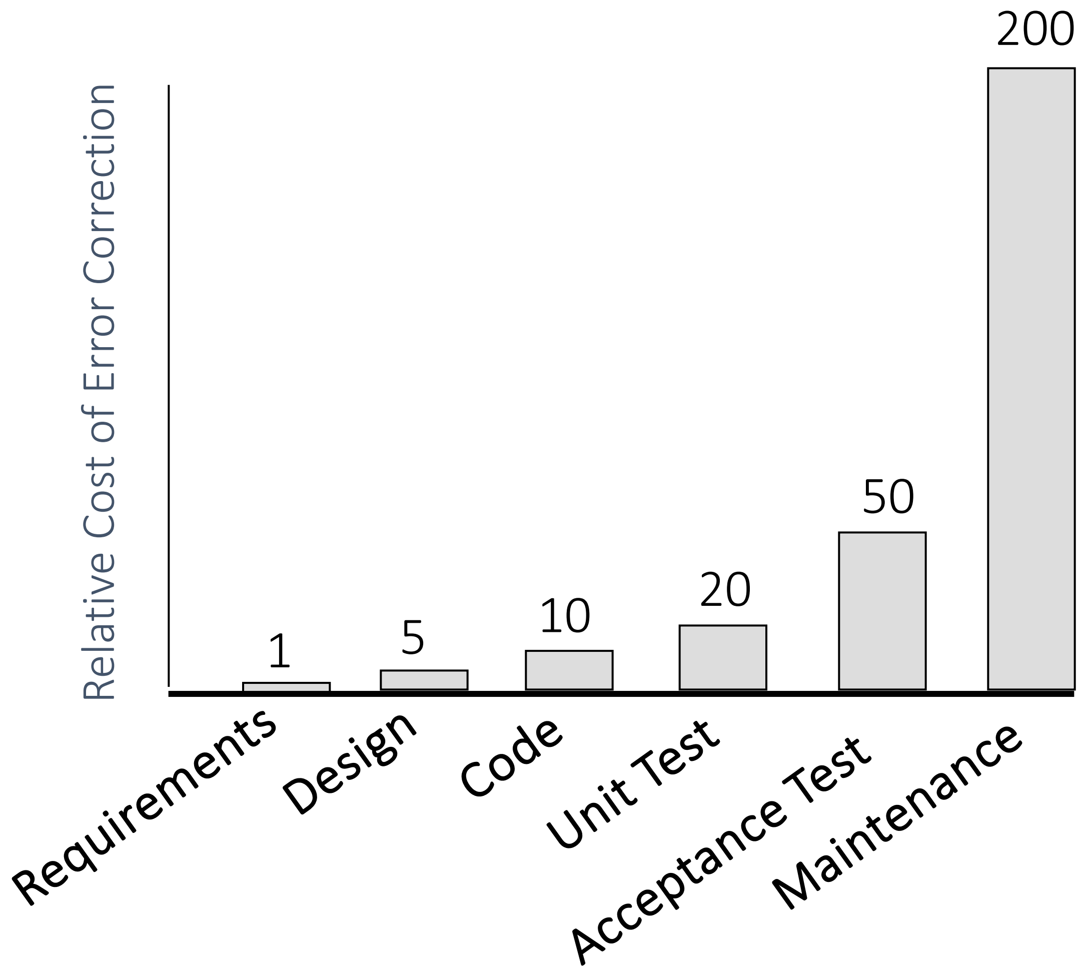
Exercise

- 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 functional, non functional and technological 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)



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

What makes RE so complex ?

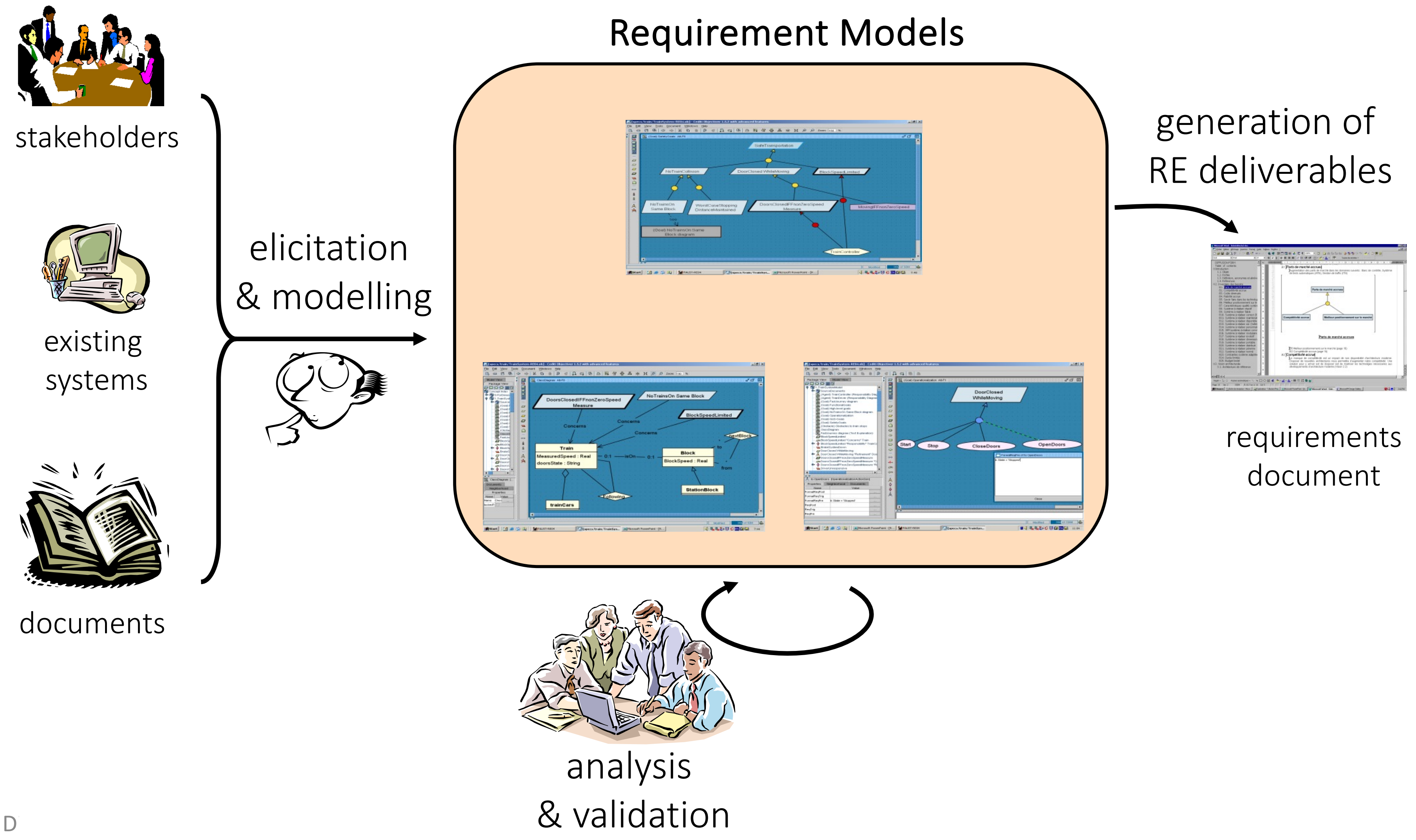
- Broad scope
  - **Composite systems**
    - human organizations + physical devices + software components
  - **More than one system**
    - system-as-is, alternative proposals for system-to-be, system evolutions, product family
  - **Multiple abstraction levels**
    - high-level goals, operational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

$\Longrightarrow$ conflicts

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

$\Longrightarrow$ conflicts

RE workflow


stakeholders
existing systems
documents
elicitation & modelling
Requirement Models
generation of RE deliverables
requirements document
analysis & validation

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 scene within 14 minutes
- 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 ambulance locations are known by GPS
- Ambulance crews correctly signal ambulance availability through mobile data terminals on board the ambulances

Examples of open questions

- Should the software system drive the ambulance?
- Who or what is the one "correctly encoding" details about incidents?
- Are mobile data terminals pre-existing or not?
- …
- Who is assigned to what?

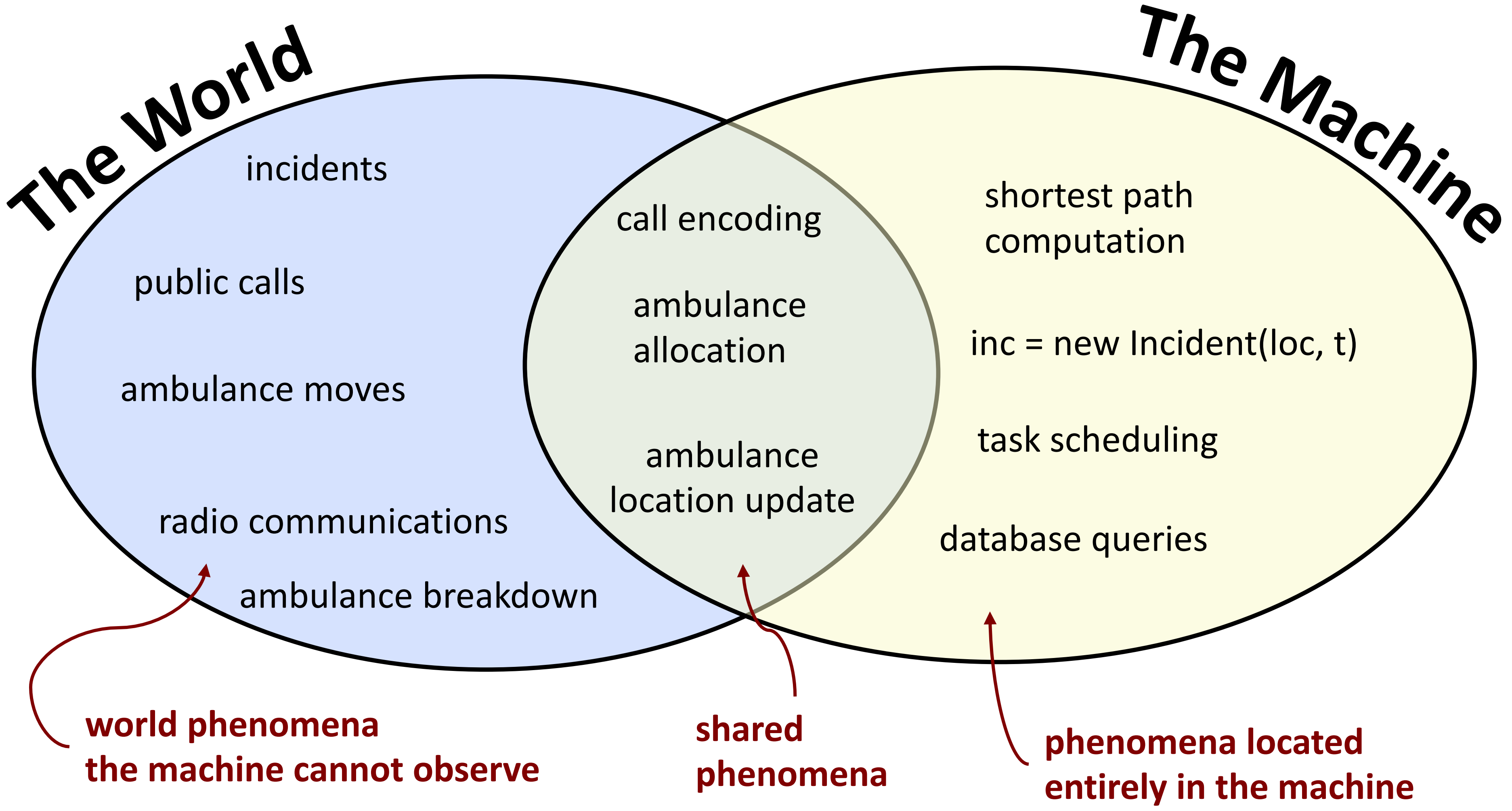
The World and the Machine
(M. Jackson & P. Zave, 1995)

- 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
- The purpose of the machine is always in the world
  - Examples
    - An Ambulance Dispatching System
    - A Banking Application
    - A Word Processor
    - …

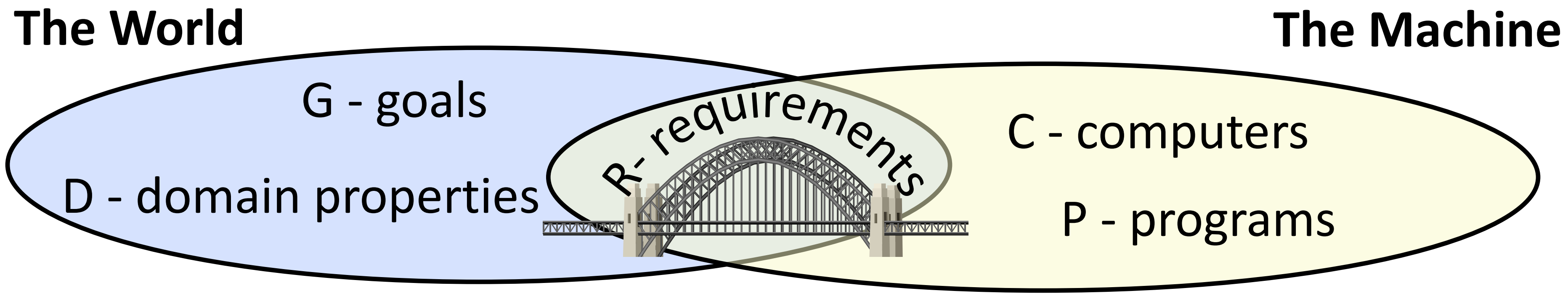
World and machine phenomena

- Requirements engineering is concerned with phenomena **occurring in the world**
  - For an ambulance dispatching system:
    - the occurrences of incidents
    - the report of incidents by public calls
    - the encodings of calls' details into the dispatching software
    - the allocation of an ambulance
    - the 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** are **prescriptive assertions** formulated in terms of world phenomena (not necessarily shared)
- **Domain properties**/assumptions are **descriptive assertions assumed to hold** in the world
- **Requirements** are **prescriptive assertions** formulated in terms of **shared** phenomena

Examples of goals, domain assumptions, 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"

Requirements' completeness?

- The requirements R are **complete** if
  1. **R** ensure satisfaction of goals **G** in the context of domain properties **D**

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

     - 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
  2. **G** adequately capture all the stakeholders' needs
  3. **D** represent valid properties/assumptions about the world

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

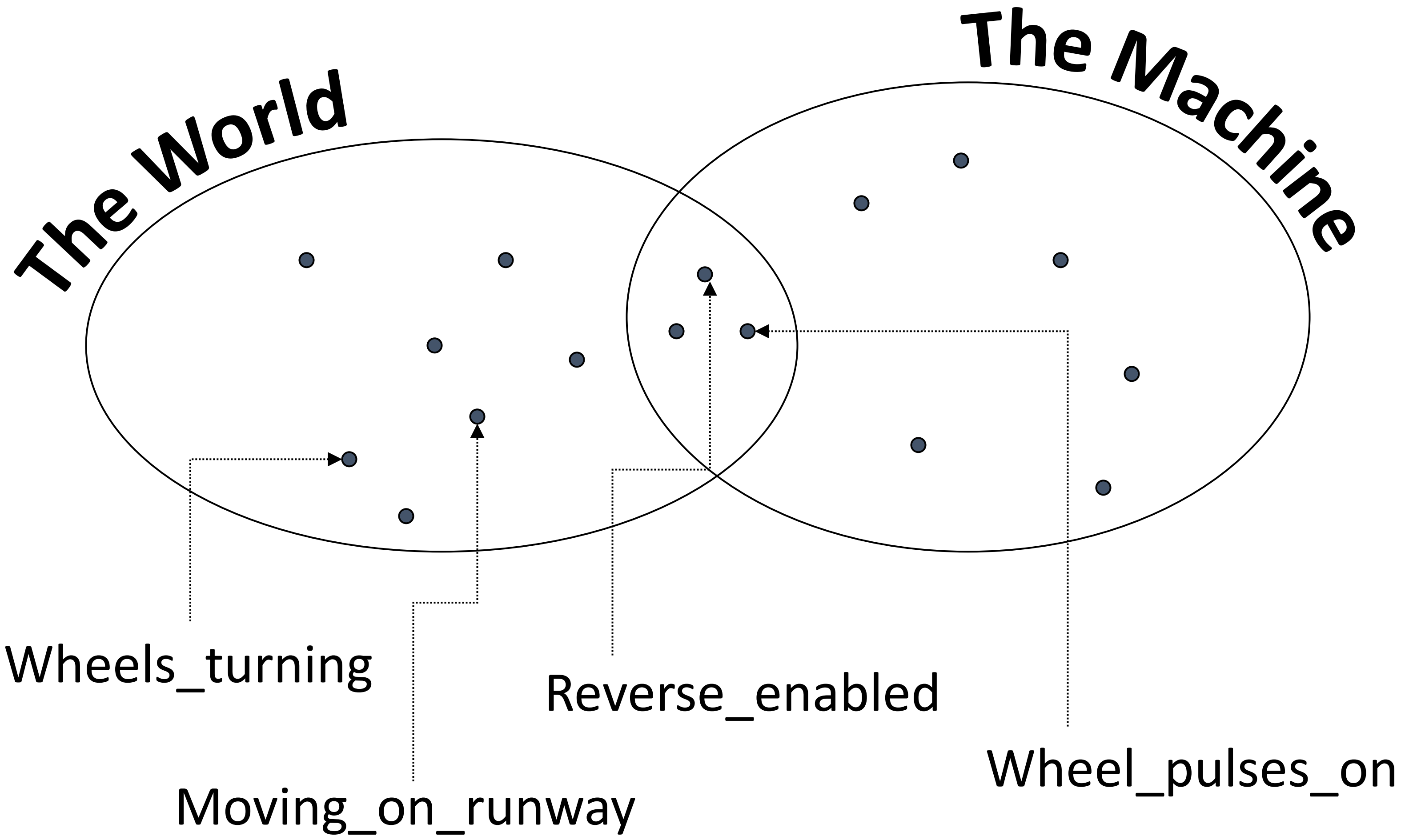
The A320 example

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)

Example – Airbus A320 Braking Logic



Goal, domain assumptions, requirement

- Goal G:
  - "Reverse thrust 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"
- Requirement R:
  - "Reverse thrust enabled if and only if wheel pulses on"

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

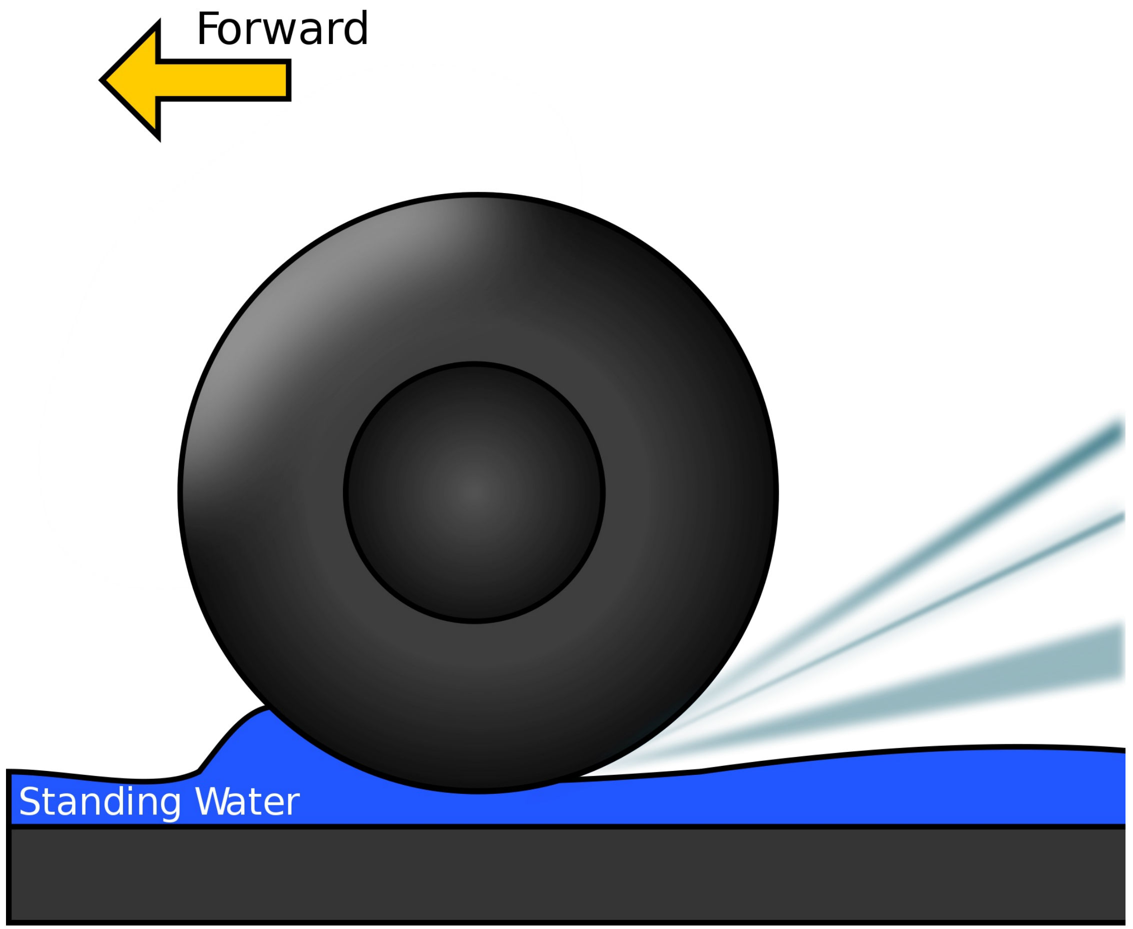
Correctness argument

- 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 $R \land D \models G$

- … but D are not valid assumptions!!!
- **Invalid domain assumptions -> Warsaw accident**

POLITECNICO
MILANO 1863

Problem: Aquaplaning



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

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
    - Watch it before Wednesday 20th
  - On modeling: https://polimi365-my.sharepoint.com/:v:/g/personal/10143828_polimi_it/EfuNIeNgPwtIrwcJQNI1s2YBw6G0vEhQDw4yNbpqLH4wQg?e=per3xm
    - Watch it before Thursday 21st
- They will be used as a basis for discussion in class next time(s)

Example of goals, assumptions, requirements

The turnstile example

The turnstile control system

- The main purpose of the system is to let people in only if they paid a coin

Domain analysis (1/2)

- Identify the relevant environment phenomena
  - here: **events**
- "Relevant" with respect to the purposes of the system
  - i.e., control people entrance

Domain analysis (2/2)

🟢	**Enter**:	person enters the room
🟢	**Push:**	person pushes turnstile
🟢🔴	**Coin**:	coin inserted
🟢🔴	**Turn**:	turnstile turns
🟡🔴	**Lock**:	$\left.\right\}$ electric signals
🟡🔴	**Unlock**:	

environment controlled

machine controlled

shared phenomena

Events and Domain Assumptions

- Events Enter, Push, Turn are clearly linked, but they are outside of the control of the machine (our control software)
- Their dependencies are captured by Domain Assumptions
  - DAs are real world properties that do not depend on the machine
- DAs linking Enter, Push, Turn:
  - D_a: Turn occurs only after a Push occurs
  - D_b: Turn leads to Enter
  - D_c: Enter occurs only after Turn occurs
- Essentially, each Enter corresponds to a Turn; on the other hand, there can be Pushes that do not lead to Turns
- We are not interested, for our application, in deeply studying the cases in which a person pushes the turnstile, but it does not turn, so 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"

Goals

G1: At any time entries should never exceed accumulated 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

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 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 compel Coin events
  - it can prevent Enter events (through Lock)

Requirements for G1

R1: Initially the turnstile is locked

R2: Command Unlock is given only if the number of pushes (p) is less than the number of accumulated payments (c)

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

R3: If Push&Turn occurs and $p = c$ holds, command Lock is given before the next Push&Turn can occur

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:

$$\begin{tikzcd} e=p \arrow[r, bend left, "Push\&Turn"] & e=p-1 \arrow[l, bend left, "Enter"] \end{tikzcd}$$

  - that is: the number of Enter (e) is ≤ the number of Push&Turn (p)
- 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 R1), so Push&Turn cannot occur (from D3)
  - Turnstile is unlocked only if $p < c$ holds (from R2), 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 must pass before a new Push&Turn can occur and (from R3) the Lock command is given before a new Push&Turn can occur
    - 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: Command Lock is given only if $p = c$ holds

R5: If $p < c$ holds and the turnstile is locked, command Unlock is given

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 D1a, 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

Observation

- The boundary between the World & the Machine is generally not given at the start of a development project
- 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.