Politecnico
di Milano

Requirements Engineering (RE)

Context

Definitions

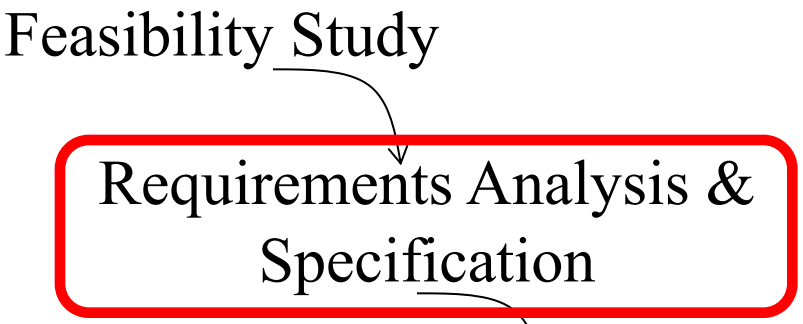
Importance and difficulties

The RE process

Understanding the world-machine relationships

The Airbus incident

Context: where do we find RE?



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*

Requirements engineering: definition

- [Jackson&Zave '95]
  - Requirements engineering is the branch of software engineering concerned with the
    - real-world goals for,
    - functions of, and
    - constraints on
  - software systems!
  - It is also concerned with the relationship of these factors to **precise specifications of software behaviour, and to their evolution over time and across software families**

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”

Functional and non-functional requirements

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
- **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 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 nonfunctional 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
- Also called **Quality of Service (QoS)** attributes

Some relevant QoS characteristics

- Performance
- Reliability
- Scalability
- Capacity
- Accuracy
- Accessibility
- Availability

Externally visible properties

- Robustness
- Exception handling

How the system works in unexpected/fault conditions

- Interoperability
- ….

Systems developed with different frameworks can work together at run time

- Integrity
- Confidentiality
- …

Security issues

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 pass or fail? Has to be fail, 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 user's 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

- Classify the following requirements in functional, non functional and technological constraints
- Highlight bad 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”

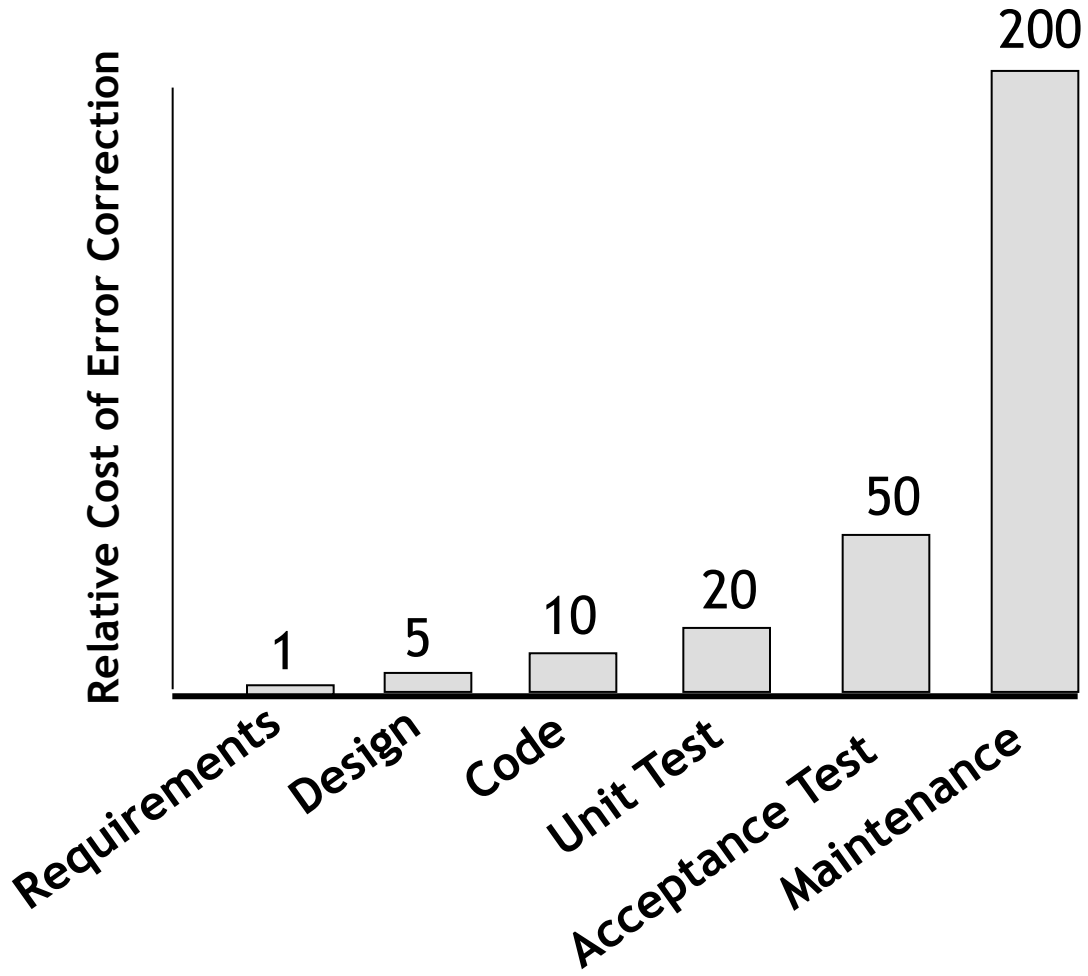
Politecnico
di Milano

Issues in RE and the requirement engineering process

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)
  - Does the list in the previous slide capture all needs of the customer?
- Prohibitive cost of late correction …
  - "up to 200 x cost of early correction” (Boehm, 1981)

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

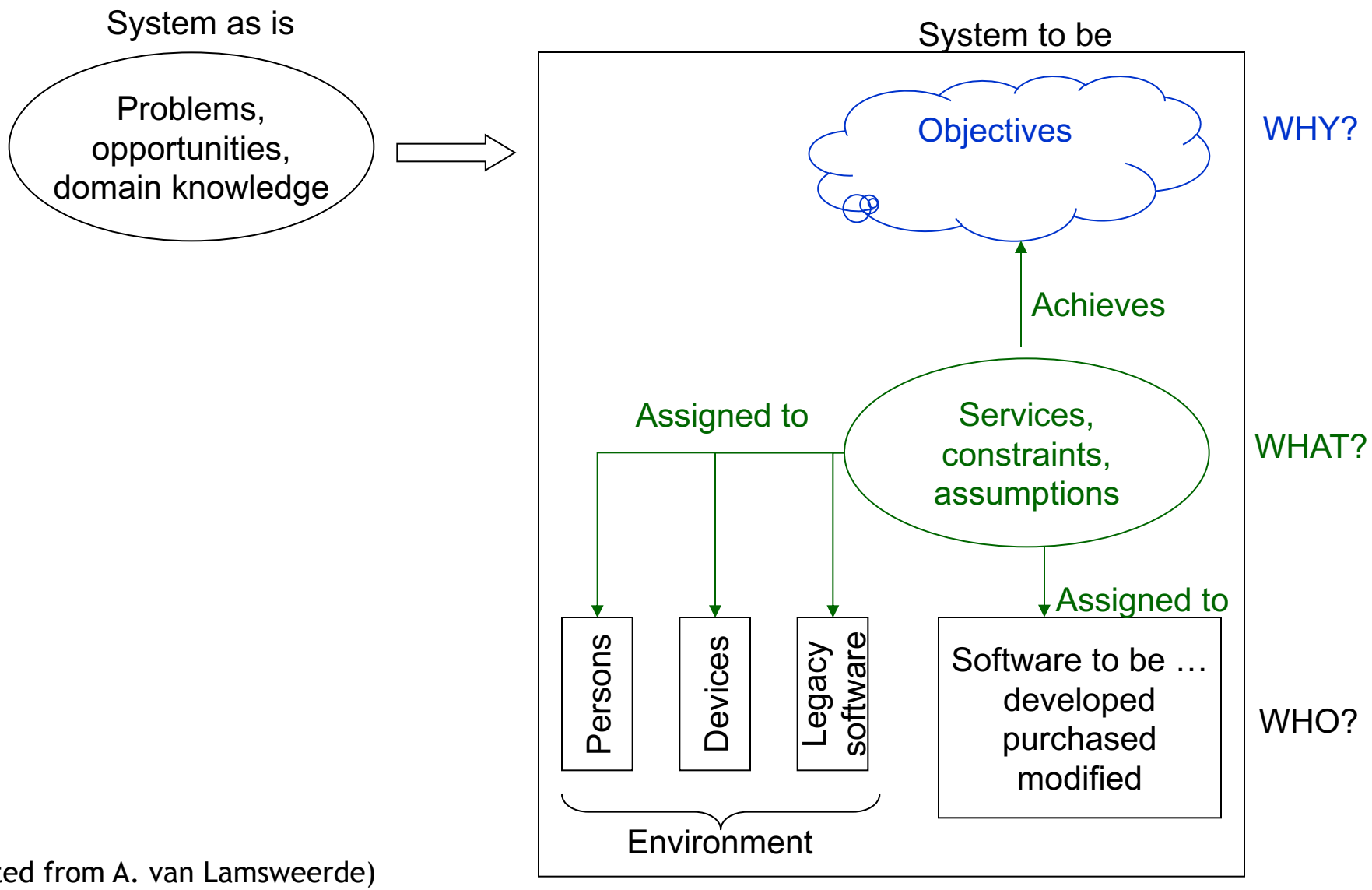
What do requirements engineers do? (1)

- Eliciting Information
  - Project objectives, context and scope
  - Domain knowledge and requirements
- Modelling and analysis
  - Goals, objects, use cases, scenarios, ...
- Communicating requirements
  - Analysis feedback, RASD document, system prototypes, ...

What do requirements engineers do? (2)

- Negotiating and agreeing Requirements
  - Handling conflicts and risks
  - Helping in requirement selection and prioritization
- Managing and evolving Requirements
  - Managing requirements during development: backward and forward traceability
  - Managing requirements changes and their impacts

The dimensions of RE



(adapted from A. van Lamsweerde)

What makes RE so complex ? (1)

- 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

(slide adapted from A. van Lamsweerde)

What makes RE so complex ? (2)

- **Multiple concerns**
  - functional, quality, development
  - hard and soft concerns

  → *conflicts*

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

  → *conflicts*

(slide adapted from A. van Lamsweerde)

The initial steps of requirement engineering

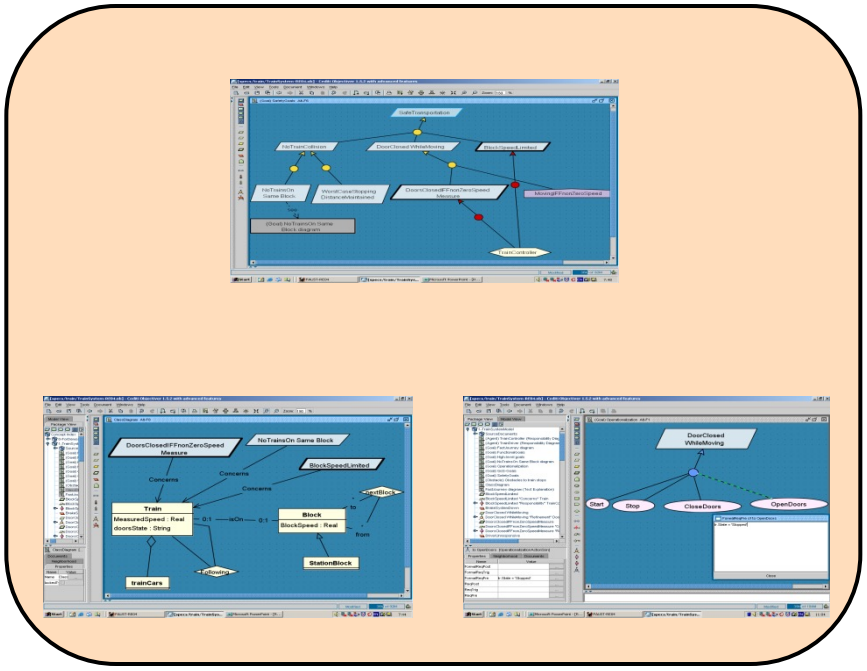
stakeholders

existing systems

documents

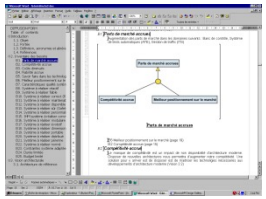
elicitation & modelling

Requirements Models



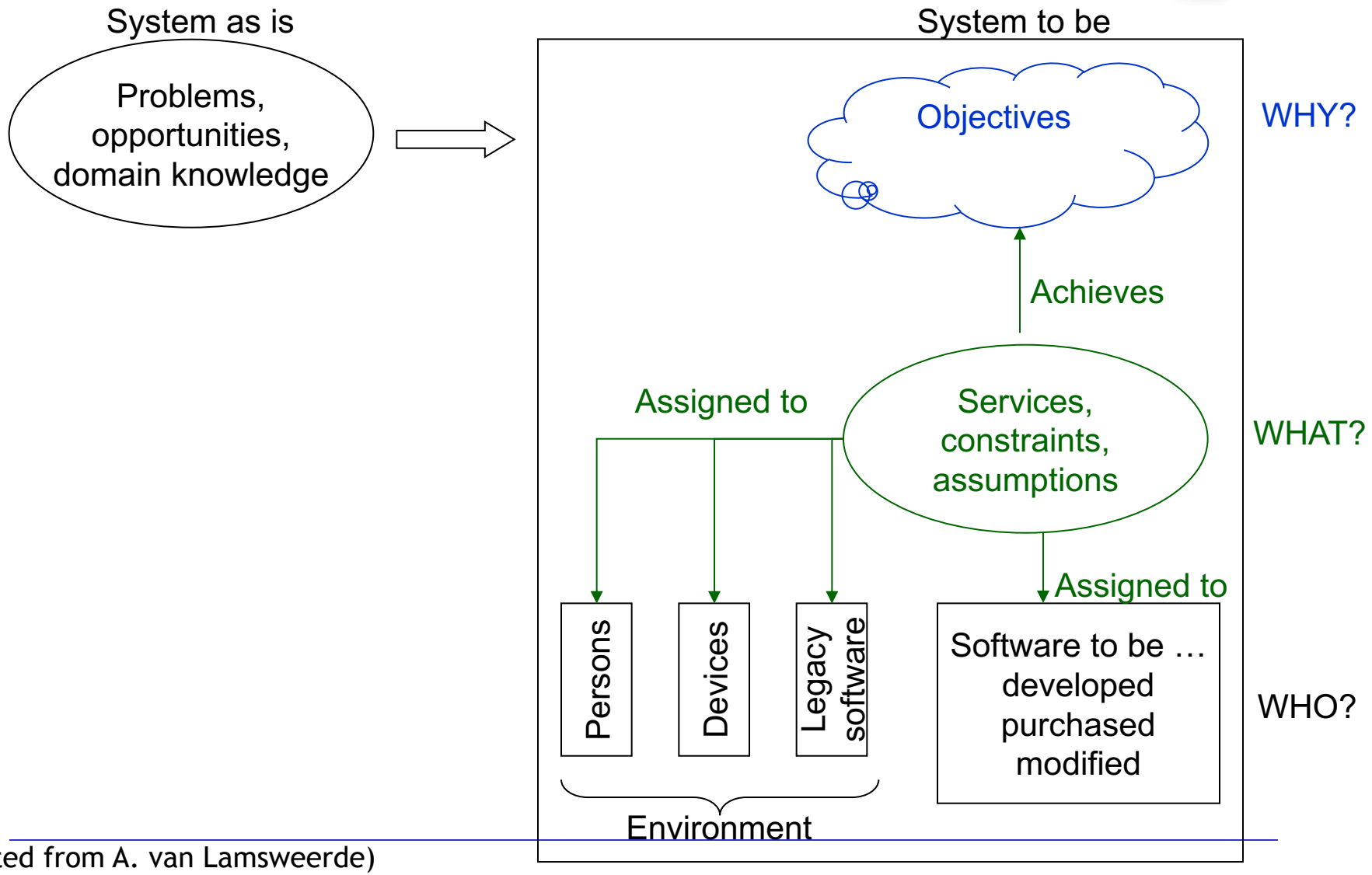
analysis & validation

generation of RE deliverables



requirements document

The starting point: requirement elicitation



(adapted from A. van Lamsweerde)

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 preexisting or not?
- …
    - Who is assigned to what?

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

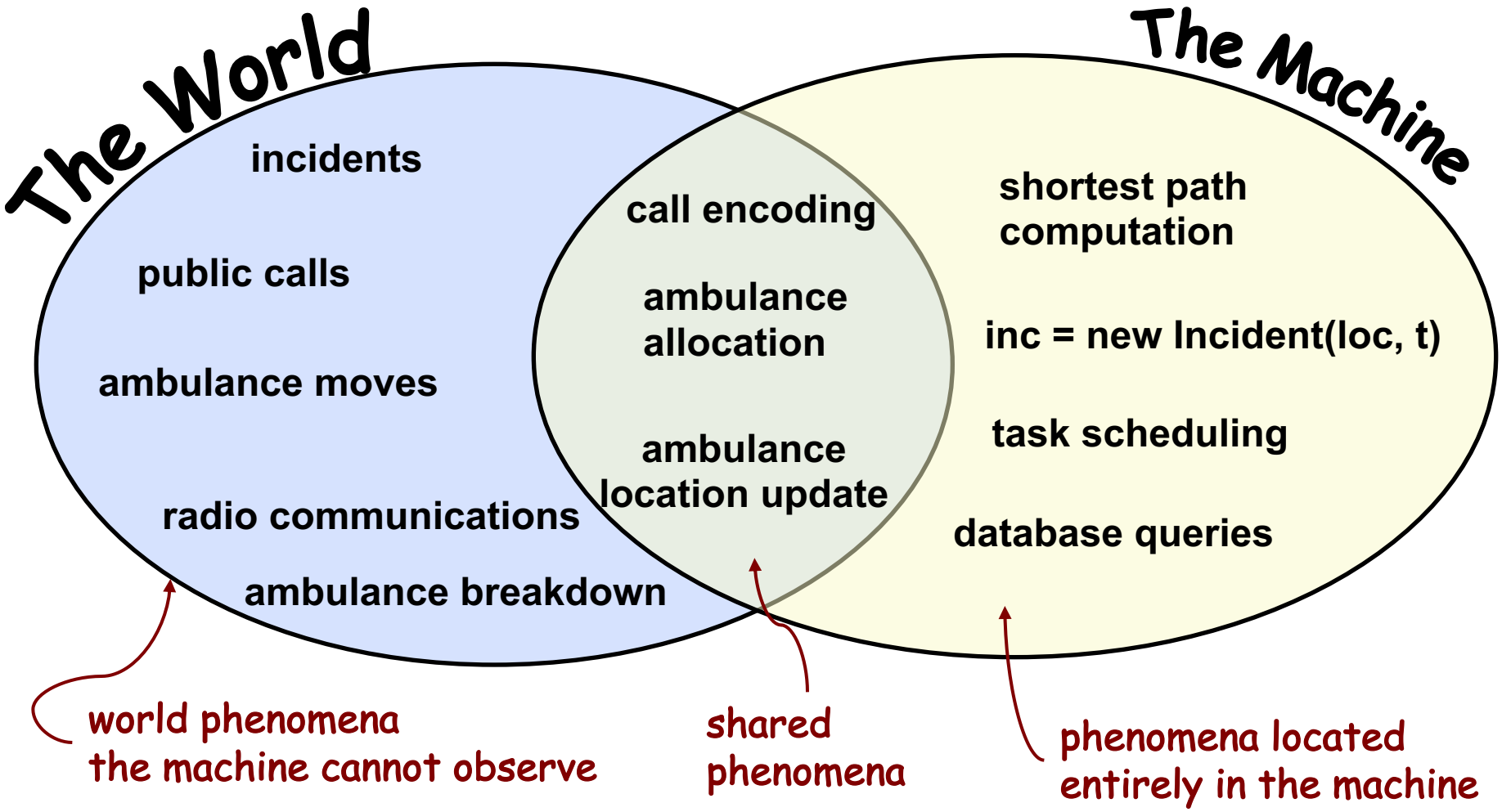
- Terminology
  - The **machine** = the portion of system to be developed
    typically, software-to-be + hardware
  - The **world** (a.k.a. the environment) = the portion of the real-world affected by the machine
- The purpose of the machine is always in the world

  Examples
  - An Ambulance Dispatching System
  - A Banking Application
  - A Word Processor
  - ...

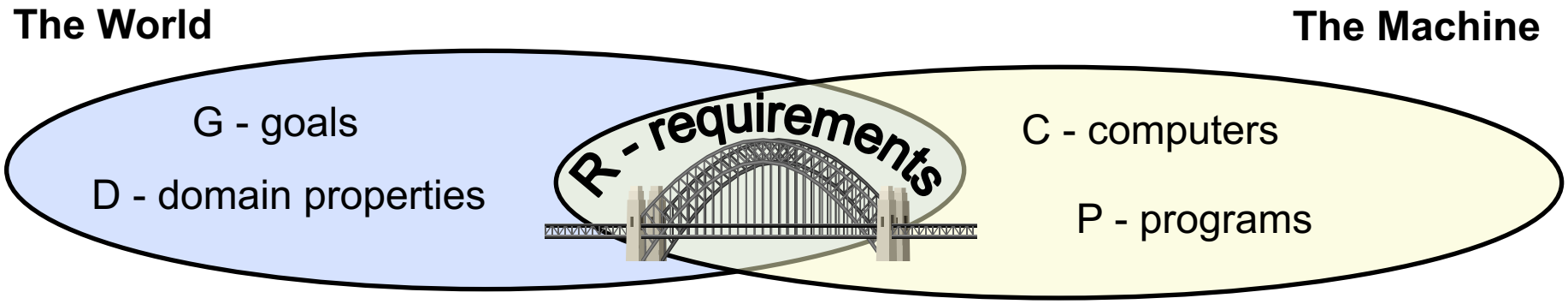
World and machine phenomena

- 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
- **Requirements models are models of the world!**

The ambulance dispatching system



Goals, domain assumptions, and 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

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*

Requirements completeness?

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

     R and D ⊨ 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 ⊨ R
  2. **G** adequately capture all the stakeholders' needs
  3. **D** represents valid properties/assumptions about the world

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

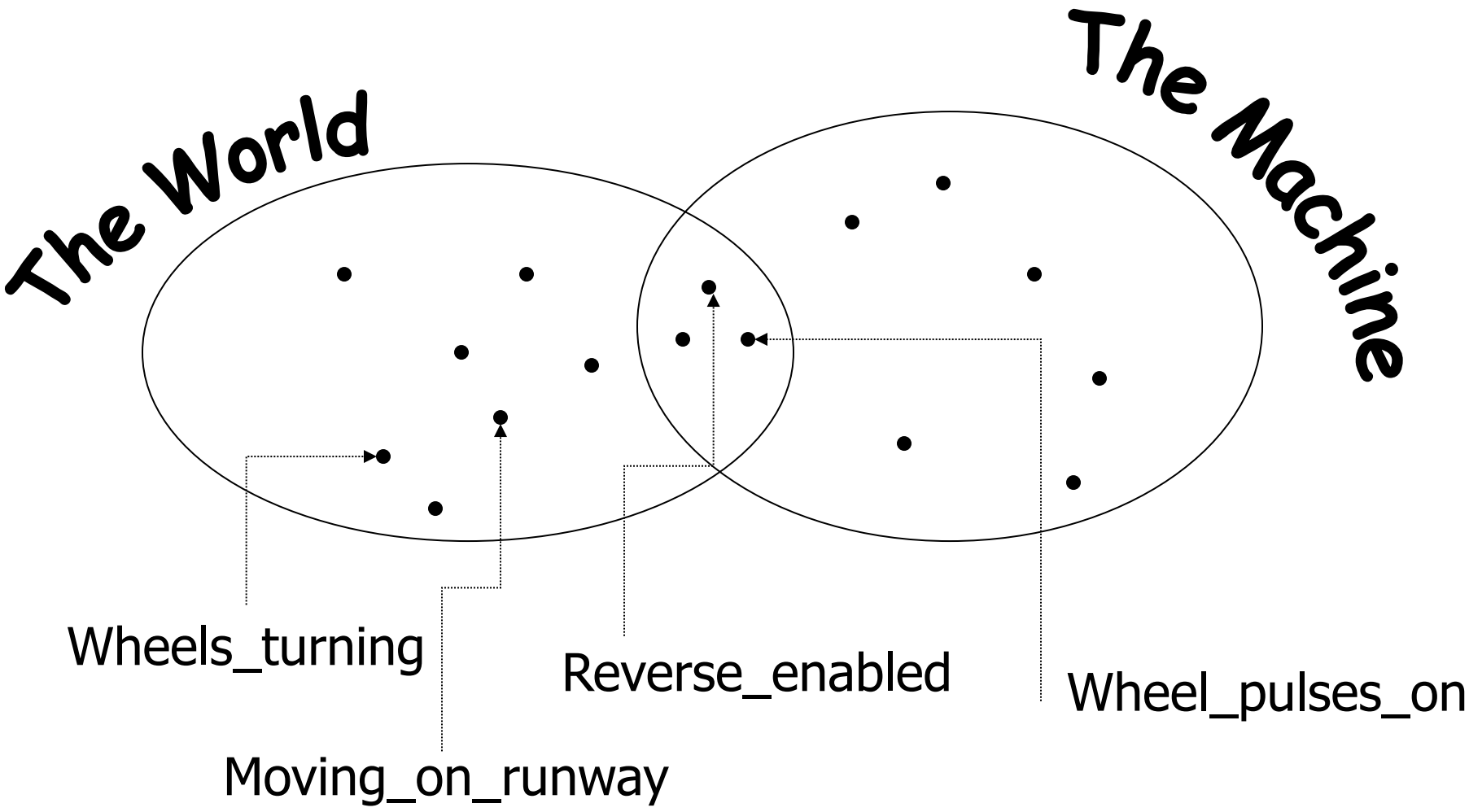
The A320 example

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)

Example – Airbus A320 Braking Logic



Goal, domain assumptions, and requirement

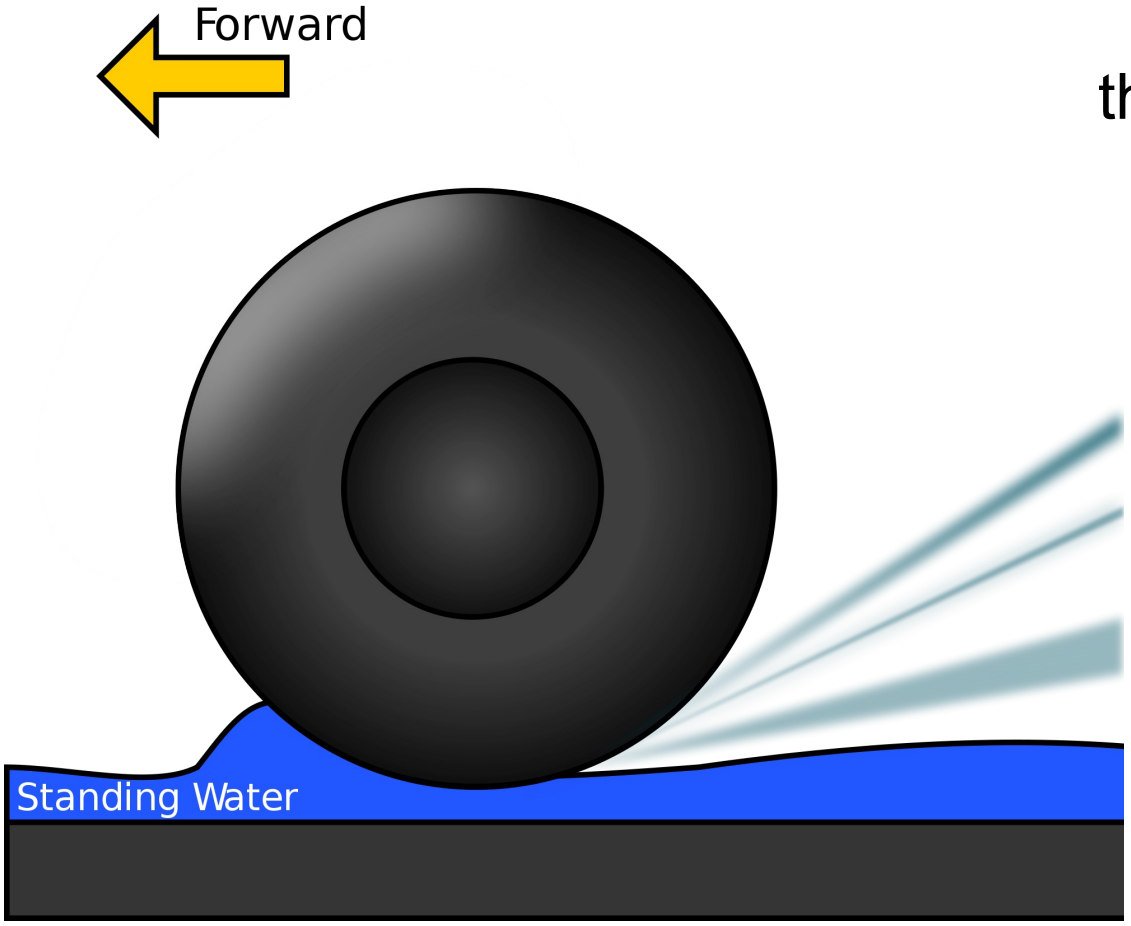
- Goal G:
  - “Reverse thrust shall be enabled if and only if the aircraft is moving on the runway”
- Domain Assumptions D:
  - Wheel pulses on if and only if wheels turning
  - Wheels turning if and only if moving on runway
- Requirements R:
  - Reverse thrust enabled if and only if wheel pulses on
- Verification: R and D $\models$ G applies!

Correctness arguments

- Goal
  - Reverse_enabled $\Leftrightarrow$ Moving_on_runway
- Domain properties
  - Wheel_pulses_on $\Leftrightarrow$ Wheels_turning
  - Wheels_turning $\Leftrightarrow$ Moving_on_runway
- Requirements
  - Reverse_enabled $\Leftrightarrow$ Wheels_pulses_on
- can prove that $R \wedge D \models G$

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

Ever heard of hydroplaning?



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

Incorrect domain assumptions lead to disasters!

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.