Requirements Engineering
- Introduction -

Slides adapted from official slides of the book “Requirements Engineering”, A. van Lamsweerde

To make sure a software solution "correctly" solves some real-world problem, we must first fully **understand** and **define** ...

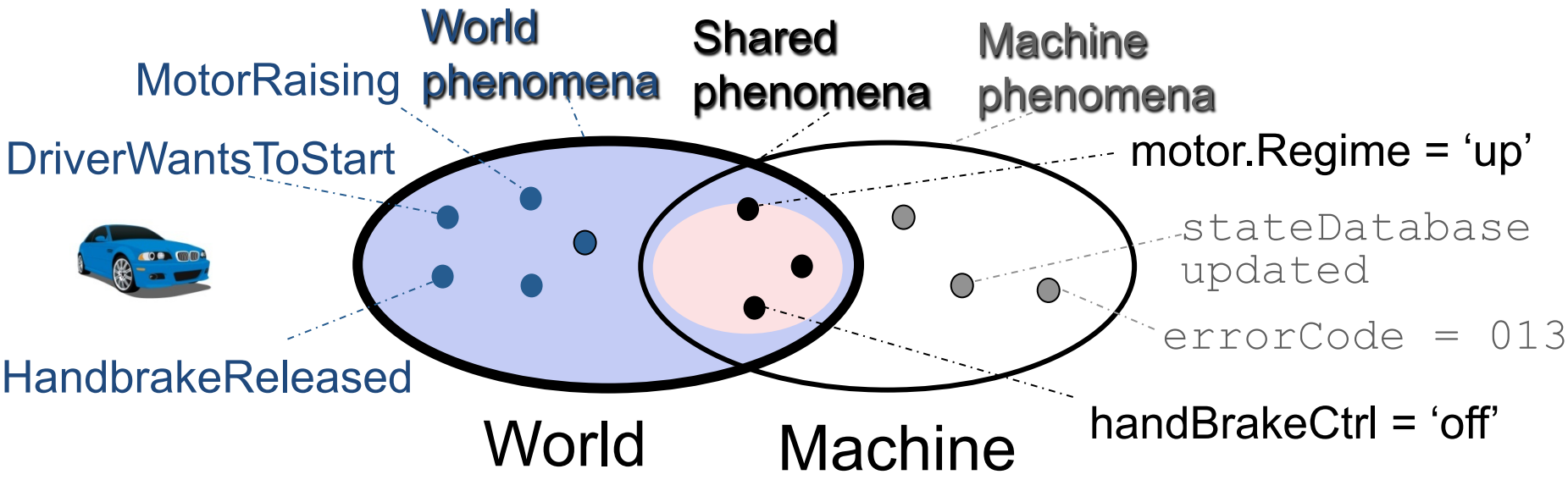
what problem needs to be solved in the real world

the **context** in which the problem arises

- Example: car control
  - **Problem**: manual handbrake release can be inconvenient in certain situations
  - **Context**: car driving, braking, driver 's intent, safety rules, …

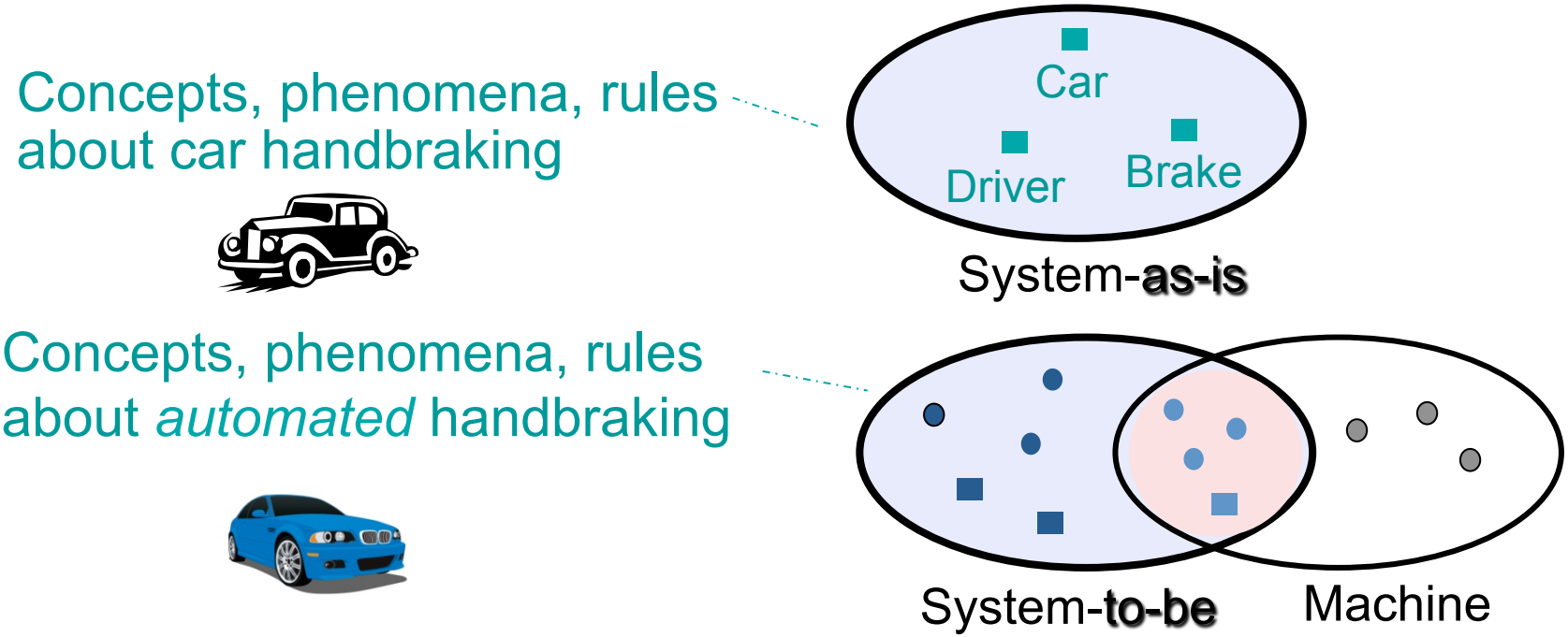
The problem world and the machine solution

- **World**: problematic part of the real-world, made of
  - human components: organization units, staff, operators, ...
  - physical components: devices, legacy software, mother Nature, ...
- **Machine**: what needs to be installed to solve the problem
  - software to be developed and/or purchased
  - hardware/software implementation platform, associated input/output devices (e.g. sensors & actuators)
- Requirements engineering (RE) is concerned with ...
  - the desired machine's **effect on the problem world**
  - the **assumptions** and **relevant properties** about this world



What does requirements engineering concern?

- System-**as-is**: system as it exists before the machine is built into it
- System-**to-be**: system as it should be when the machine will operate into it



RE: a preliminary definition

Coordinated set of **activities** ...

- for **exploring**, **evaluating**, **documenting**, **consolidating**, **revising** and **adapting** the **objectives**, **capabilities**, **qualities**, **constraints** & **assumptions** on a software-intensive **system**
- based on **problems** raised by the system-**as-is** and **opportunities** provided by new technologies

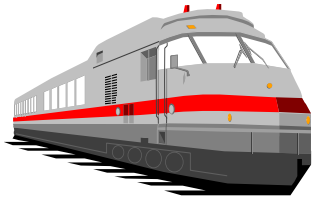
What others said ...

Ross'77

- Requirements definition must say ...
  - **why** a new system is needed, based on current or foreseen conditions,
  - **what** system features will satisfy this context,
  - **how** the system is to be constructed

Zave'97

- RE is concerned with the real-world **goals** for, **functions** of, **constraints** on software systems; and with their
  - **link** to precise **specifications of software behavior**,
  - **evolution** over time and families



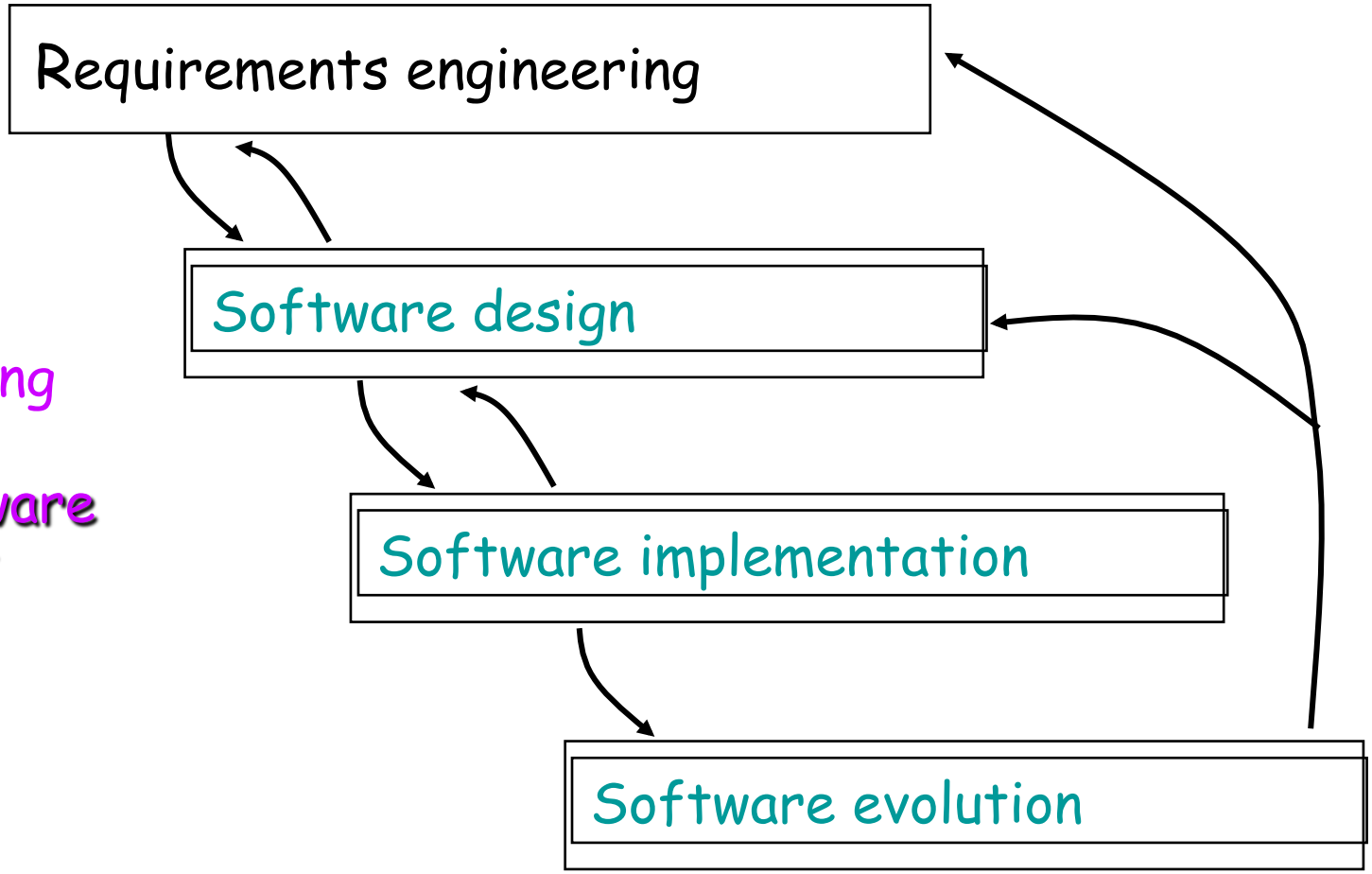
Example: transportation between airport terminals

- Problem (system-**as-is**):
  - passengers frequently missing flight connections among different terminals; slow & inconvenient transportation
  - number of passengers regularly increasing
- Objectives, options (system-**to-be**):
  - support high-frequency trains between terminals
  - with **or** without train drivers ?
- Functionalities, constraints:
  - software-based control of train accelerations, doors opening etc. to achieve *prompt* and *safe* transportation
- RE deliverable: requirements document for system-to-be

Requirements in the software lifecycle

Getting the *right* system

Getting the **software** *right*



What makes RE hard ?

- Broad scope
  - multiple system versions: *as-is*, *to-be, to-be-next*
  - hybrid environment: human organizations, policies, regulations, devices, physical laws
- Multiple concerns
  - functional, quality, development concerns
- Multiple abstraction levels
  - strategic objectives, operational details
- Multiple stakeholders
  - with different background, different interests and conflicting viewpoints
- Multiple intertwined tasks during iterative elicitation-evaluation-specification-consolidation
  - conflict management, risk management, evaluation of alternatives, prioritization, quality assurance, change anticipation

System requirements vs. **software** requirements

System requirements: what the *system*-to-be should meet; formulated in terms of phenomena in the environment

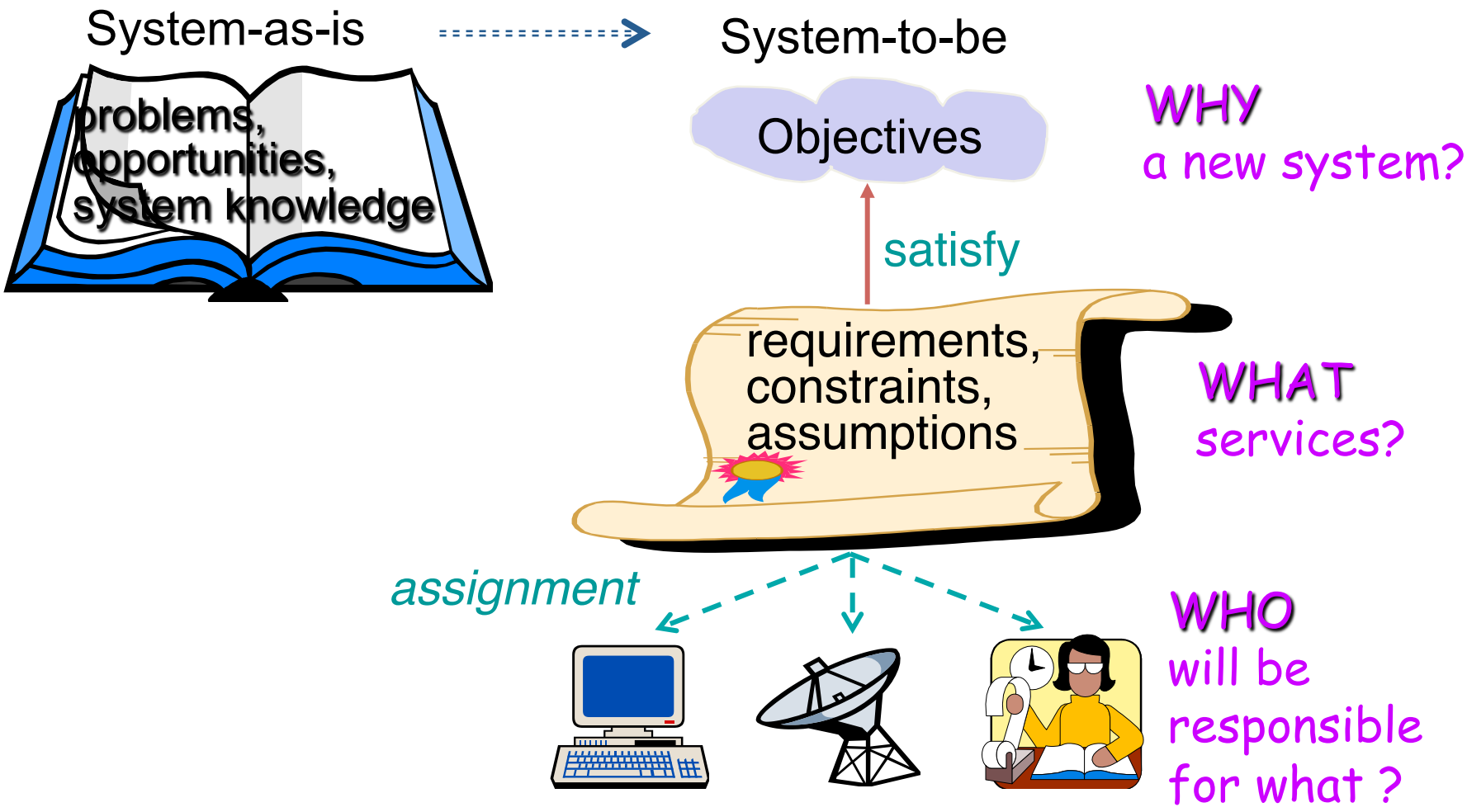
"The handbrake shall be released when the driver wants to start."

Software requirements: what the *software*-to-be should meet on its own; formulated in terms of phenomena **shared** by the software and the environment

"The software output variable *handBrakeCtrl* shall have the value *off* when the software input variable *motorRegime* gets the value *up*."

System = software + environment
(people, devices, other software)

The scope of RE: the *WHY, WHAT, WHO* dimensions



The WHY dimension

- Identify, analyze, refine the system-to-be's **objectives**
  - to address analyzed deficiencies of the system-as-is
  - in alignment with business objectives
  - taking advantage of technology opportunities
- Example: airport train control
  "Serve more passengers"
  "Reduce transfer time among terminals"
- Difficulties
  - Acquire domain knowledge
  - Evaluate alternative options (e.g., alternative ways of satisfying the same objective)
  - Match problems-opportunities, and evaluate these: implications, associated risks
  - Handle conflicting objectives

The WHY dimension

- Identify, analyze, refine the system-to-be's **objectives**
  - to address analyzed deficiencies of the system-as-is
  - in alignment with business objectives
  - taking advantage of technology opportunities
- Example: airport train control
  "Serve more passengers"
  "Reduce transfer time among terminals"
- Difficulties
  - Acquire domain knowledge
  - Evaluate alternative options (e.g., alternative ways of satisfying the same objective)
  - Match problems-opportunities, and evaluate these: implications, associated risks
  - Handle conflicting objectives

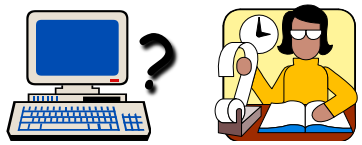
Do you already know ways of evaluating alternative options?

The WHAT dimension

- Identify & define the system-to-be's **functional services** (software services, associated manual procedures)
  - to satisfy the identified **objectives**
  - according to **quality constraints**: security, performance, ...
  - based on realistic **assumptions** about the environment

- Example: airport train control
  "Computation of safe train accelerations"
  "Display of useful information for passengers inside trains"

- Difficulties
  - Identify the right set of features
  - Specify these precisely for understanding by all parties
  - Ensure backward traceability to system objectives



The WHO dimension

- Assign responsibilities for the objectives, services, constraints among system-to-be components
  - based on their capabilities and on the system's objectives
  - yielding the software-environment boundary

- Example: airport train control
  - "Safe train acceleration" ... under direct responsibility of software-to-be (driverless option) ***or*** of driver following software indications ?
  - "Accurate estimation of train speed/position" ... under responsibility of tracking system ***or*** of preceding train ?

- Difficulties
  - Evaluate alternative options to decide on the right degree of automation

Question

What are the “Why”, the “What” and the “Who” of the SIFA system?

Types of Requirements

Types of Statements

Functional vs Non-Functional

Statements may differ in mood

What is the difference between

The same copy cannot be borrowed by two different people at the same time.

A patron may not borrow more than three books at the same time.

A person cannot physically attend two meetings on different continents on the same day.

The meeting date must fit the constraints of all important participants.

If train doors are open, they are not closed.

Train doors shall always remain closed when the train is moving.

Statements may differ in mood

Descriptive statements (indicative mood)

The same copy cannot be borrowed by two different people at the same time.

A person cannot physically attend two meetings on different continents on the same day.

If train doors are open, they are not closed.

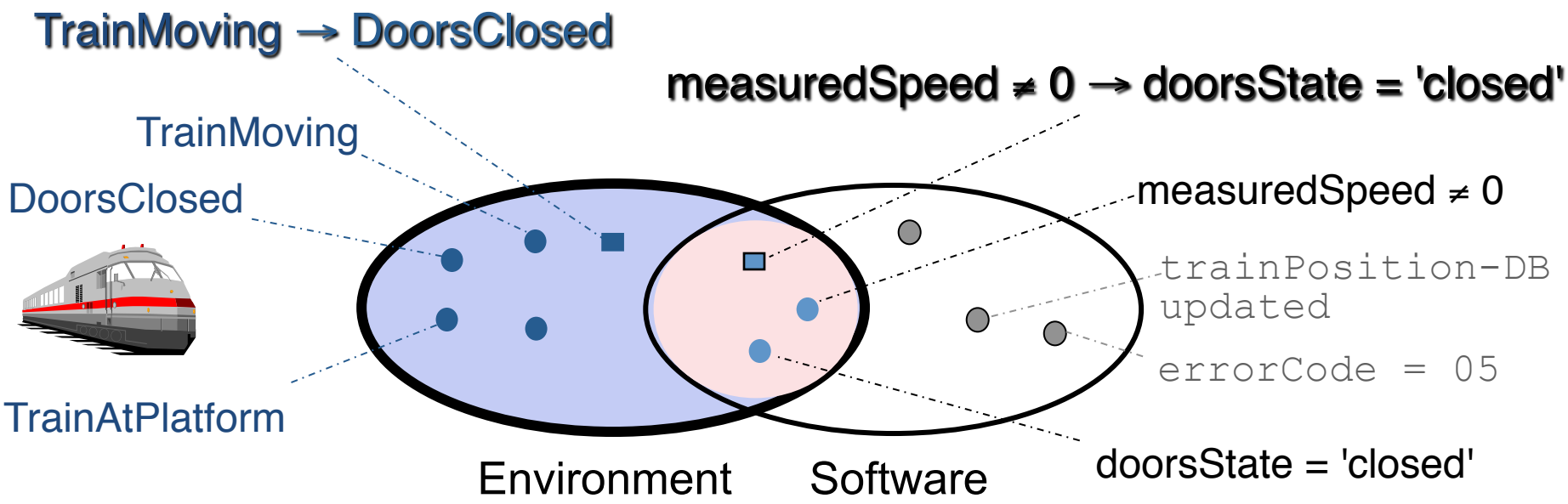
Prescriptive statements (optative mood)

A patron may not borrow more than three books at the same time.

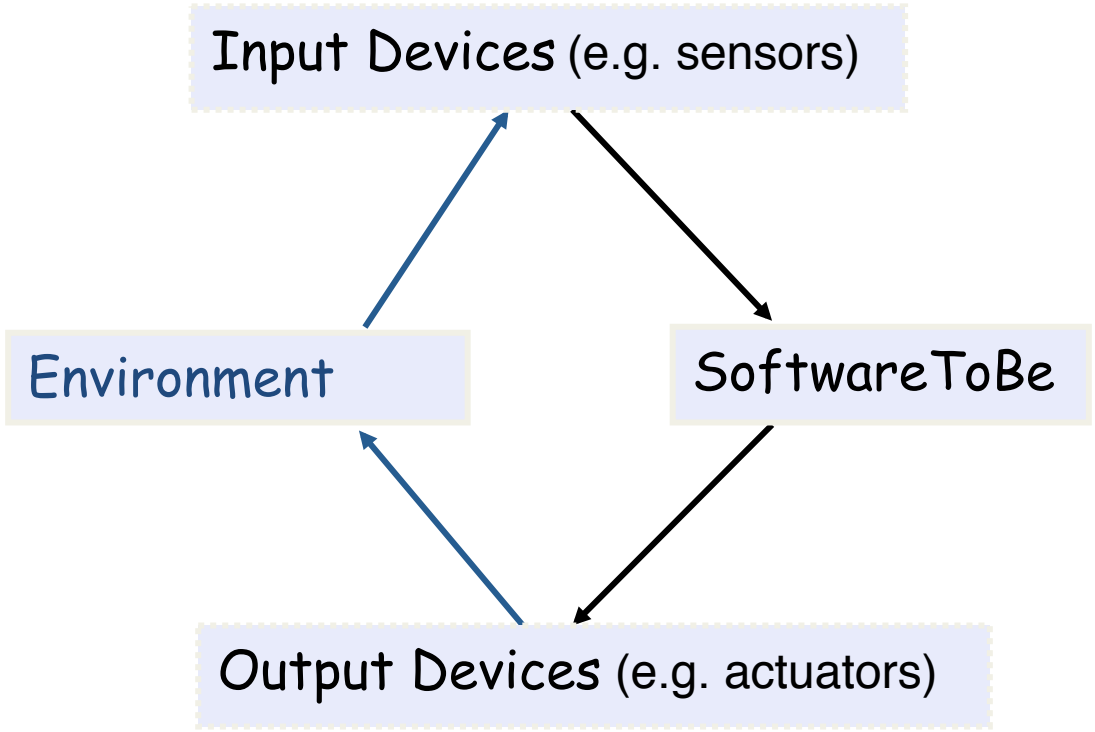
The meeting date must fit the constraints of all important participants.

Train doors shall always remain closed when the train is moving.

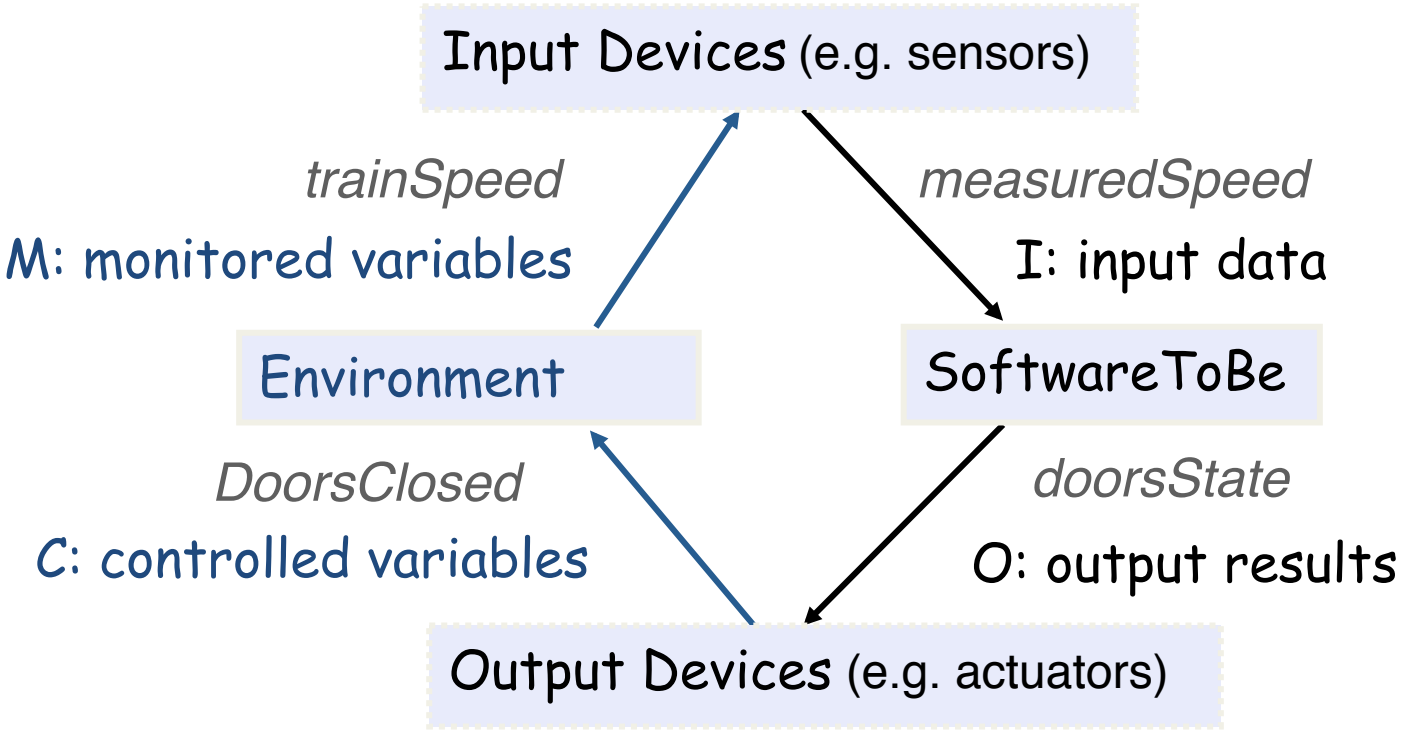
Statements may differ in scope



The 4-variable model [Parnas95]



The 4-variable model [Parnas95]



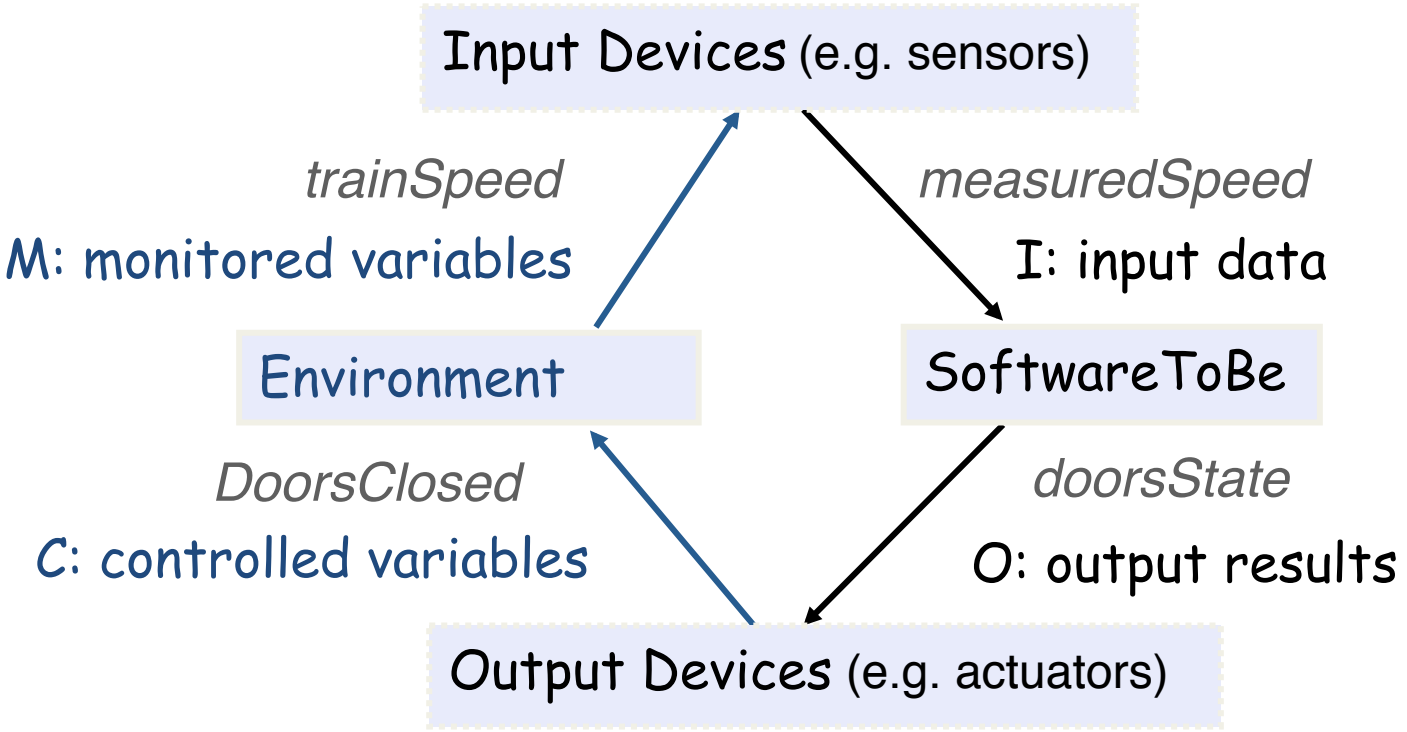
Types of statements

- **System requirement**: *prescriptive* statement referring to *environment* phenomena (not necessarily shared)
  - enforced by the software-to-be possibly together with other system components
  - understandable by all parties

All train doors shall always remain closed while a train is moving.

Patron may not borrow more than three books at time.

The 4-variable model [Parnas95]



What is a system requirement in the 4-variable model?

SysReq $\subseteq M \times C$ relation on environment monitored/controlled variables

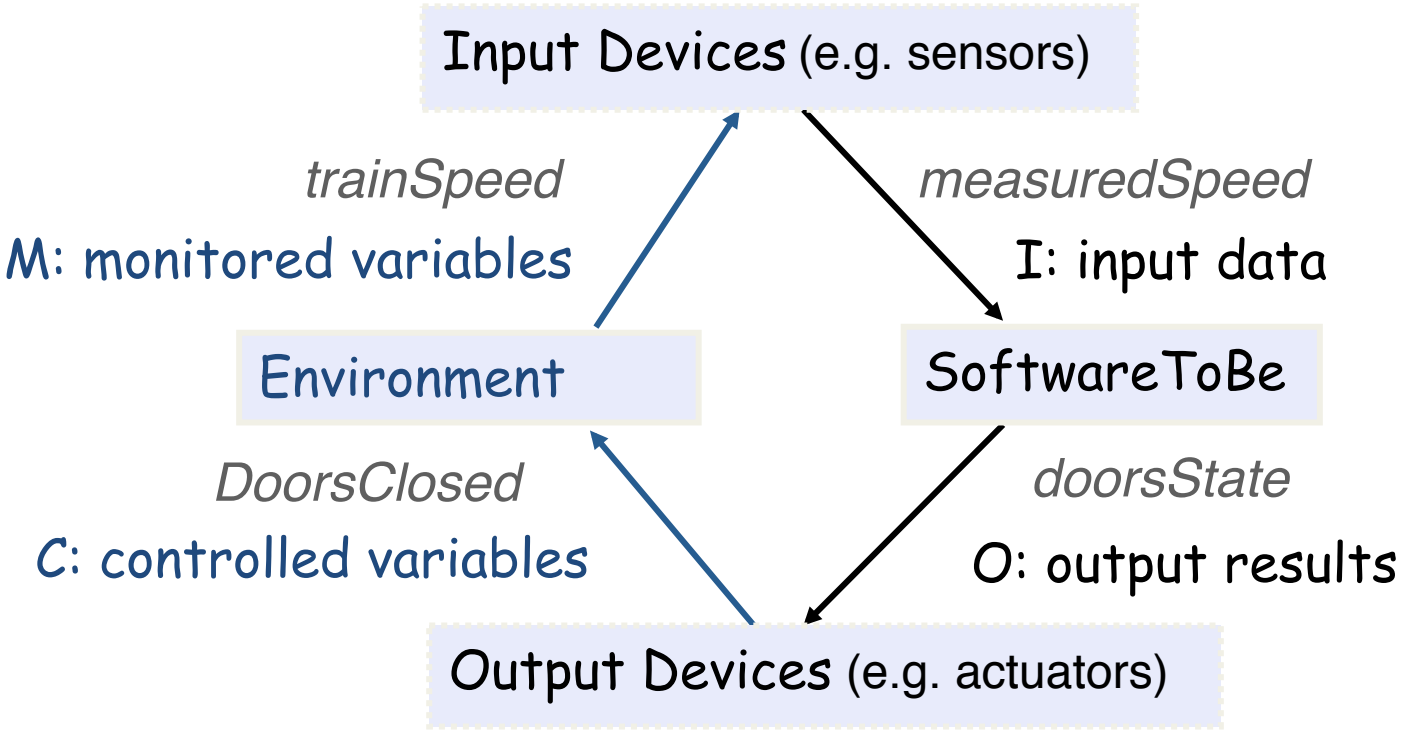
Types of statements

- **Software requirement**: *prescriptive* statement referring to *shared* phenomena
  - enforced by the software-to-be solely
  - formulated in the vocabulary of software developers

The doorState output variable shall always have the value ‘closed’ when the measuredSpeed input variable has a non-null value.

The recorded number of loans by a patron may never exceed a maximum number x.

The 4-variable model [Parnas95]



What is a software requirement in the 4-variable model?

SofReq $\subseteq I \times O$ relation on software input/output variables

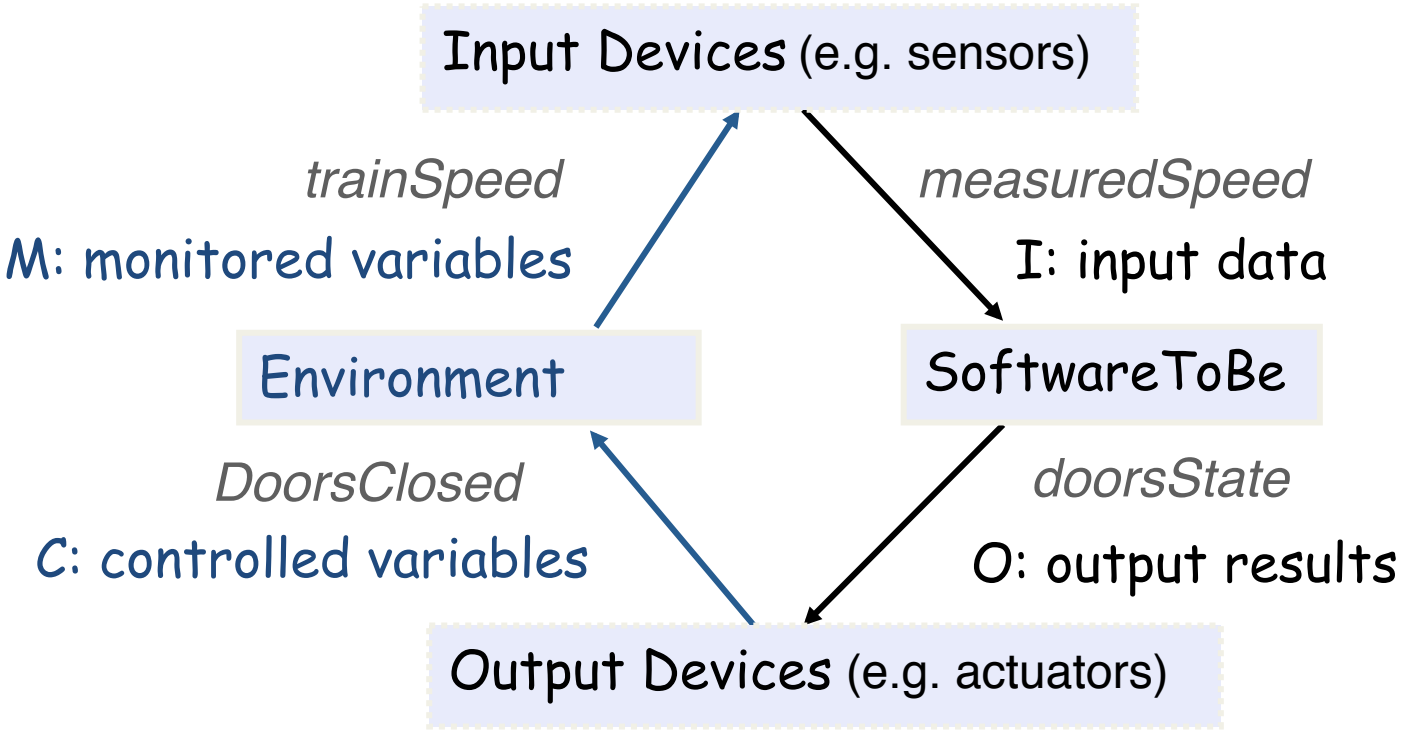
Types of statements

- **Domain property**: *descriptive* statement about problem world phenomena (holds regardless of any software-to-be)

a train is moving if and only if its physical speed is non-null.

a book may not be borrowed and available at the same time.

The 4-variable model [Parnas95]



What is a domain property in the 4-variable model?

Dom $\subseteq M \times C$ laws that cannot be broken

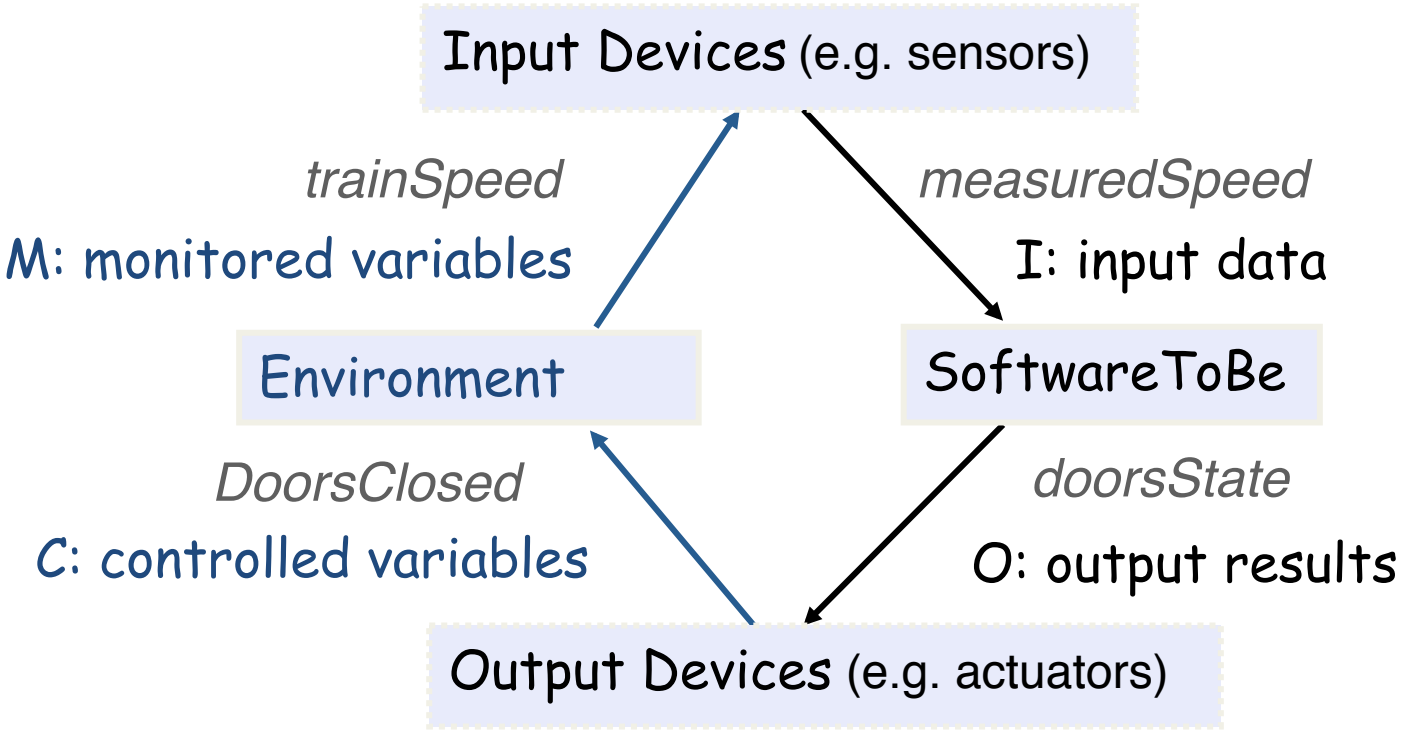
Types of statements

- **Assumption**: statement to be satisfied by the environment of the software-to-be
  - formulated in terms of environment phenomena
  - generally prescriptive (e.g., on sensors or actuators)

A train's measured speed is non-null if and only if its physical speed is non-null.

The recorded number of loans by a borrower is equal to the actual number of book copies physically borrowed by him or her.

The 4-variable model [Parnas95]



What is an assumption in the 4-variable model?

$Asm \subseteq M \times C \;\cup\; M \times I \;\cup\; C \times O$

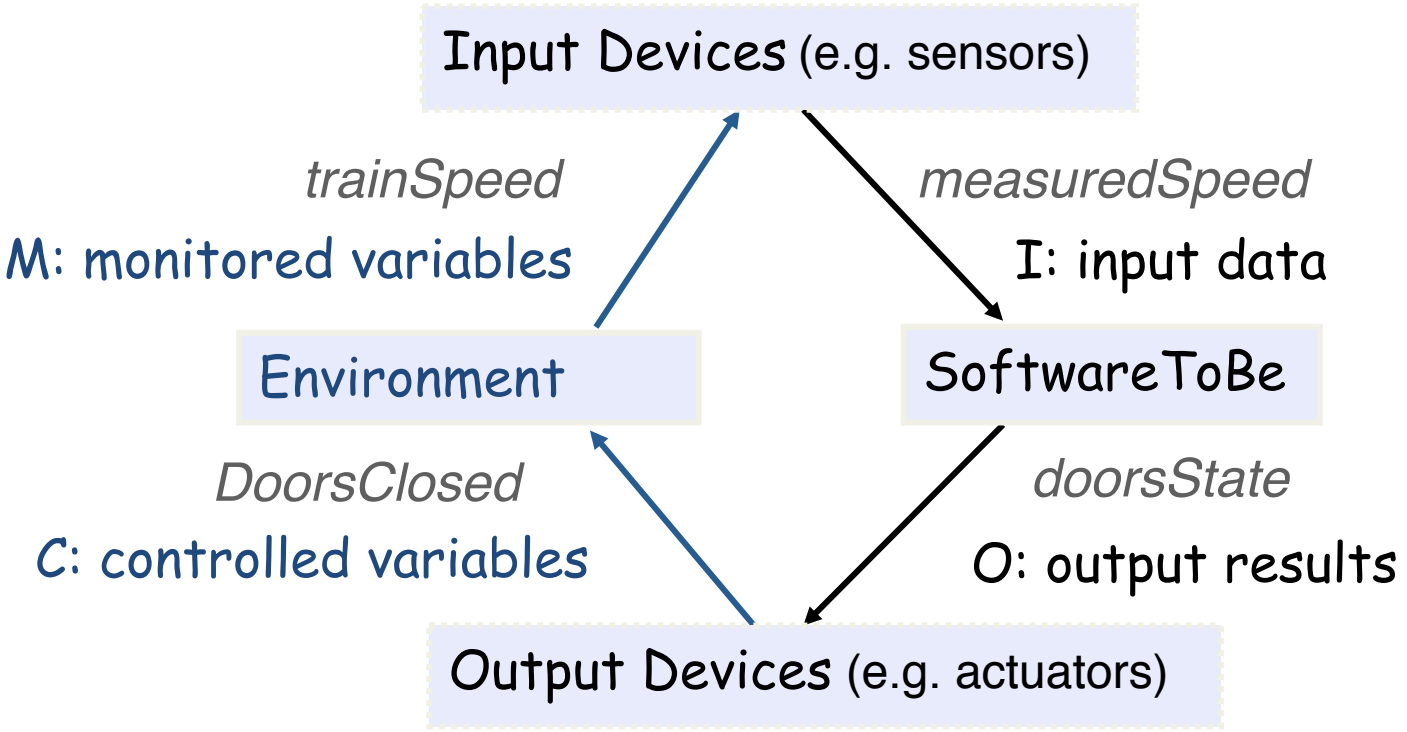
Types of statements

- **Definition**: statement providing a precise meaning to system concepts or auxiliary terms
  - no truth value
    - It does not make sense to question about satisfaction of a definition
    - … but accuracy, completeness, adequacy, etc… still apply

measuredSpeed is the speed estimated by the train's speedometer.

a person participates in a meeting if he or she attends that meeting from beginning to end.

Relating System and Software Requirements



SofReq = *Map* (SysReq, Dom, Asm)

translates SysReq using domain properties and assumptions

Mapping system reqs to software reqs involves satisfaction arguments

SOFREQ, ASM, DOM |= *SysReq*

"**If** the software requirements in SOFREQ, the assumptions in ASM and the domain properties in DOM are all satisfied and consistent, **then** the system requirements *SysReq* are satisfied"

(sofReq) measuredSpeed≠0 ➔ doorsState=‘closed’

The doorState output variable shall always have the value ‘closed’ when the measuredSpeed input variable has a non-null value.

All train doors shall always remain closed while a train is moving.

(sysReq) TrainMoving ➔ DoorsClosed

(sofReq) measuredSpeed≠0 ➔ doorsState=‘closed’

(Asm) measuredSpeed≠0 ⬅➔ trainSpeed≠0

A train’s measured speed is non-null if and only if its physical speed is non-null.

measuredSpeed≠0 ➔ doorsState=‘closed’

trainSpeed≠0 ➔ doorsState=‘closed’

(sysReq) TrainMoving ➔ DoorsClosed

(sofReq) measuredSpeed≠0 ➔ doorsState=‘closed’

(Asm) measuredSpeed≠0 ⬅➔ trainSpeed≠0

(Dom) TrainMoving ⬅➔ trainSpeed≠0

a train is moving if and only if its physical speed is non-null.

trainSpeed≠0 ➔ doorsState=‘closed’

TrainMoving ➔ doorsState=‘closed’

(sysReq) TrainMoving ➔ DoorsClosed

(sofReq) measuredSpeed≠0 ➔ doorsState=‘closed’

(Asm) measuredSpeed≠0 ⬅➔ trainSpeed≠0

(Dom) TrainMoving ⬅➔ trainSpeed≠0

(Asm) doorsState=‘closed’⬅➔ DoorsClosed

A train’s doors are closed if and only if the doors are recognized as closed

TrainMoving ➔ doorsState=‘closed’

(sysReq) TrainMoving ➔ DoorsClosed

System requirements cannot be satisfied by software requirements only!

We need to elicit, evaluate, document, consolidate relevant assumptions & domain properties

Example: inadequate domain property in A320 braking logic

SofReq: reverse = 'on' **iff** WheelPulses = 'on'

ASM: reverse = 'on' **iff** ReverseThrustEnabled

WheelPulses = 'on' **iff** WheelsTurning

Dom: MovingOnRunway **iff** WheelsTurning

SysReq: ReverseThrustEnabled **iff** MovingOnRunway

Example: inadequate domain property in A320 braking logic

SofReq: reverse = 'on' **iff** WheelPulses = 'on'

ASM: reverse = 'on' **iff** ReverseThrustEnabled

WheelPulses = 'on' **iff** WheelsTurning

Dom: ~~MovingOnRunway **iff** WheelsTurning~~

SysReq: ReverseThrustEnabled **iff** MovingOnRunway

Warsaw crash: plane moving on waterlogged runway with no wheels turning (aquaplaning)

Categories of requirements

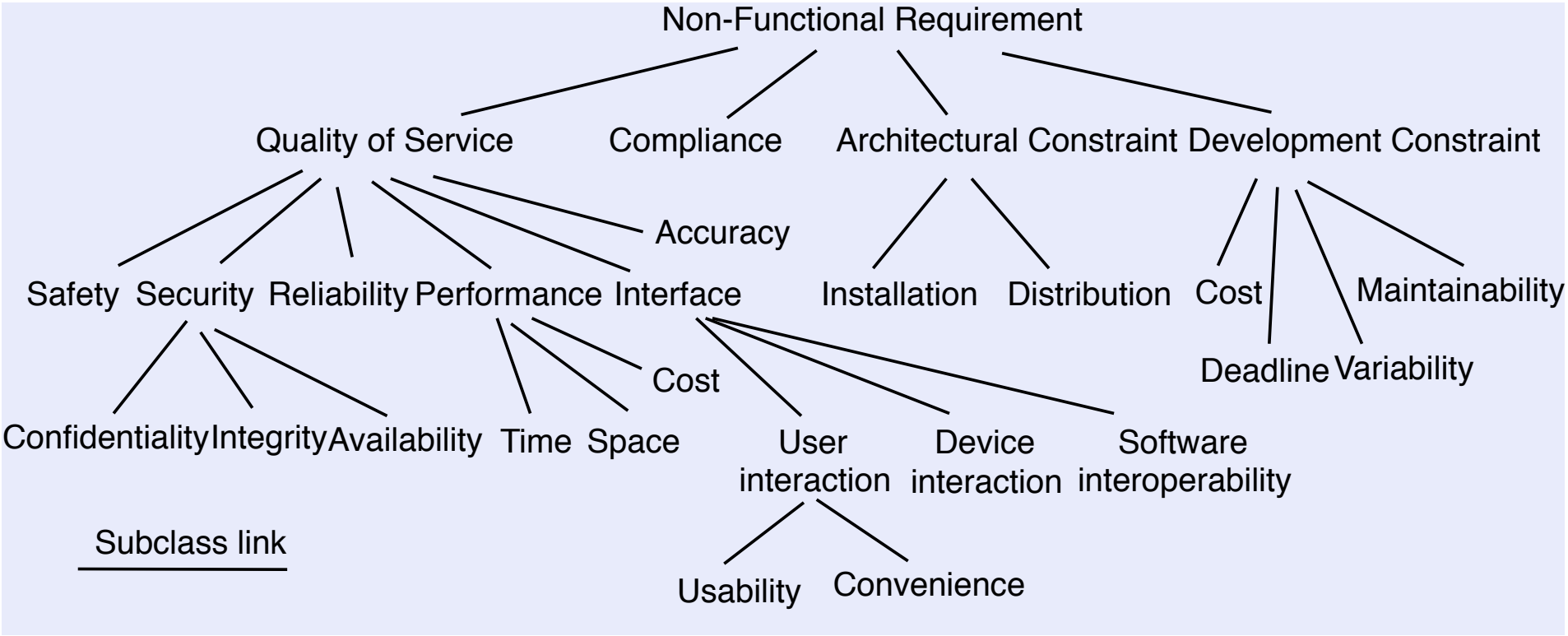
- **Functional requirements:** prescribe what services the software-to-be should provide

"The software shall control the acceleration of all trains"

- **Non-functional requirements:** constrain how such services should be provided

"Acceleration commands shall be issued every 3 seconds to every train"

A taxonomy of non-functional requirements



- No clear-cut boundaries, possible overlaps
  - Functional/non-functional: e.g., functional reqs for firewall management are security-related
  - Non-functional overlaps: e.g., "high frequency of train commands" is related to performance and safety

Requirements Qualities

Target qualities for RE process

- **Completeness** of objectives, requirements, assumptions
- **Consistency** of RD items
- **Adequacy** of requirements, assumptions, domain props
- **Unambiguity** of RD items
- **Measurability** of requirements, assumptions
- **Pertinence** of requirements, assumptions
- **Feasibility** of requirements
- **Comprehensibility** of RD items
- **Good structuring** of the RD
- **Modifiability** of RD items
- **Traceability** of RD items

Errors in a requirements document

- **Omission**: problem world feature not stated by any RD item
  e.g., no req about state of train doors in case of emergency stop
- **Contradiction**: RD items stating a problem world feature in an incompatible way
  "Doors must always be kept closed between platforms"
  and "Doors must be opened in case of emergency stop"
- **Inadequacy**: RD item not adequately stating a problem world feature
  "if a book…not been returned, the negligent borrower shall be notified that he or she has to pay a fine"
- **Ambiguity**: RD item allowing a problem world feature to be interpreted in different ways
  "Only the people who attended at least 75% of the meetings could be awarded at the end of the year"
- **Unmeasurability**: RD item stating a problem world feature in a way precluding option comparison or solution testing
  "Panels inside trains shall be user-friendly"

Flaws in a requirements document

- **Noise**: RD item yielding no information on any problem world feature (Variant: uncontrolled redundancy)
  "Non-smoking signs shall be posted on train windows"
- **Overspecification**: RD item stating a feature not in the problem world, but in the machine solution
  "The *setAlarm* method shall be invoked on receipt of an *Alarm* message"
- **Unfeasibility**: RD item not implementable within budget/schedule
  "In-train panels shall display all delayed flights at next stop"
- **Unintelligibility**: RD item incomprehensible to those needing to use it
  "In the US, the notion of an NWO became popular after the terrorist attacks on the WTC. However, officials in NATO and the WTO rarely refer to an NWO in proceedings relating to the GATT, and it can be said that the MVTO, the MFN clause, and SROs have little to do with an NOW" (from a press release)
- **Poor structuring**: RD item not organized according to any sensible & visible structuring rule
  Intertwining of acceleration control and train tracking issues

Flaws in a requirements document

- **Forward reference**: RD item making use of problem world features not defined yet

  Multiple uses of the concept of *worst-case stopping distance* before its definition appears several pages after in the RD

- **Remorse**: RD item stating a problem world feature lately or incidentally

  After multiple uses of the undefined concept of *worst-case stopping distance*, the last one directly followed by an incidental definition between parentheses

- **Poor modifiability**: RD items whose changes must be propagated throughout the RD

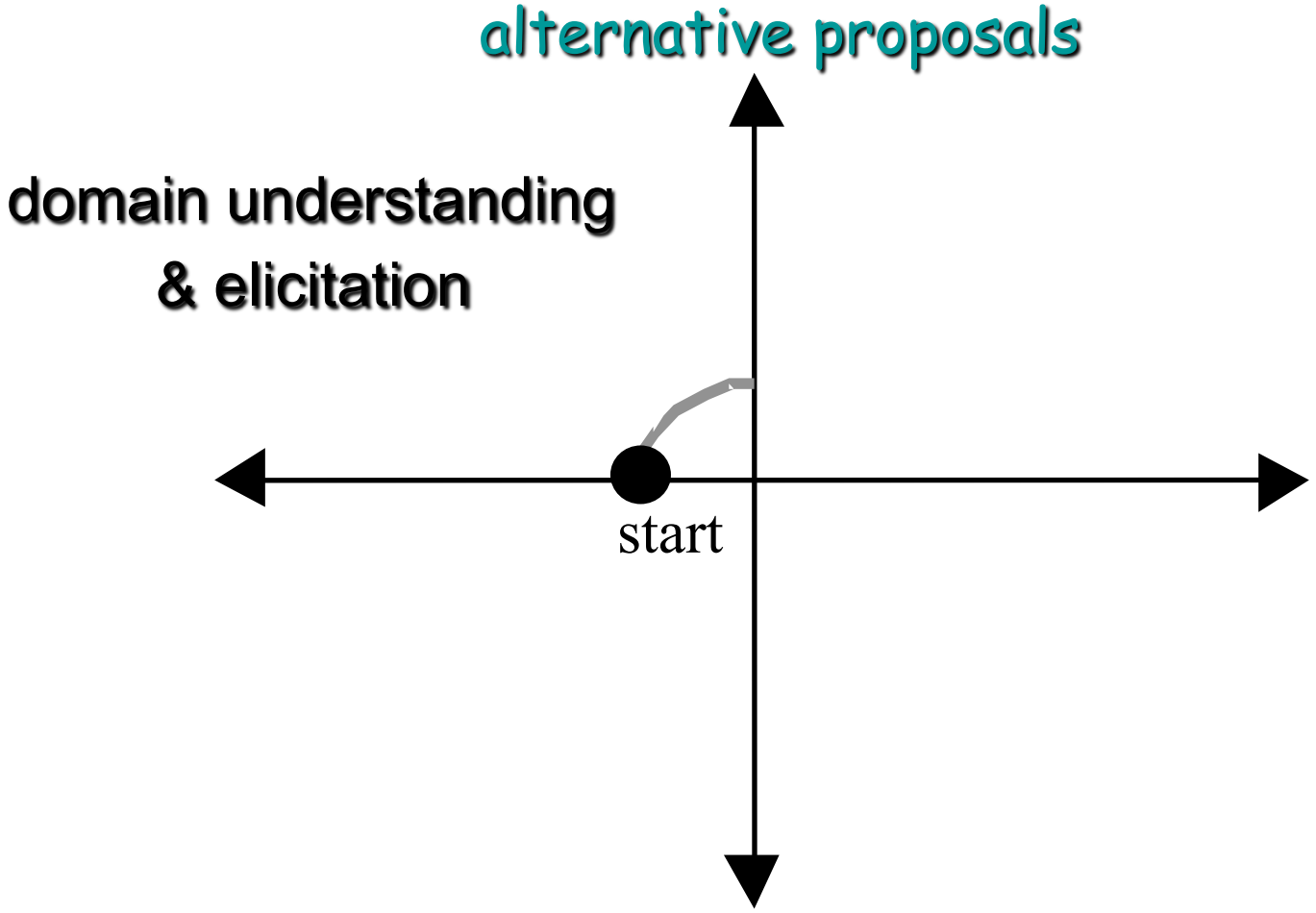
  Use of fixed numerical values for quantities subject to change

- **Opacity**: RD item whose rationale, authoring or dependencies are invisible

  "The commanded train speed must always be at least 7 mph above physical speed" *without* any explanation of rationale for this

The Requirements Engineering Process

The RE process



Domain understanding

- Studying the system-as-is
  - Business organization
  - Application domain
  - Strengths & weaknesses of the system-as-is

- Identifying the system **stakeholders**

- **Result**:
  - Initial sections for preliminary draft proposal
  - Glossary of terms

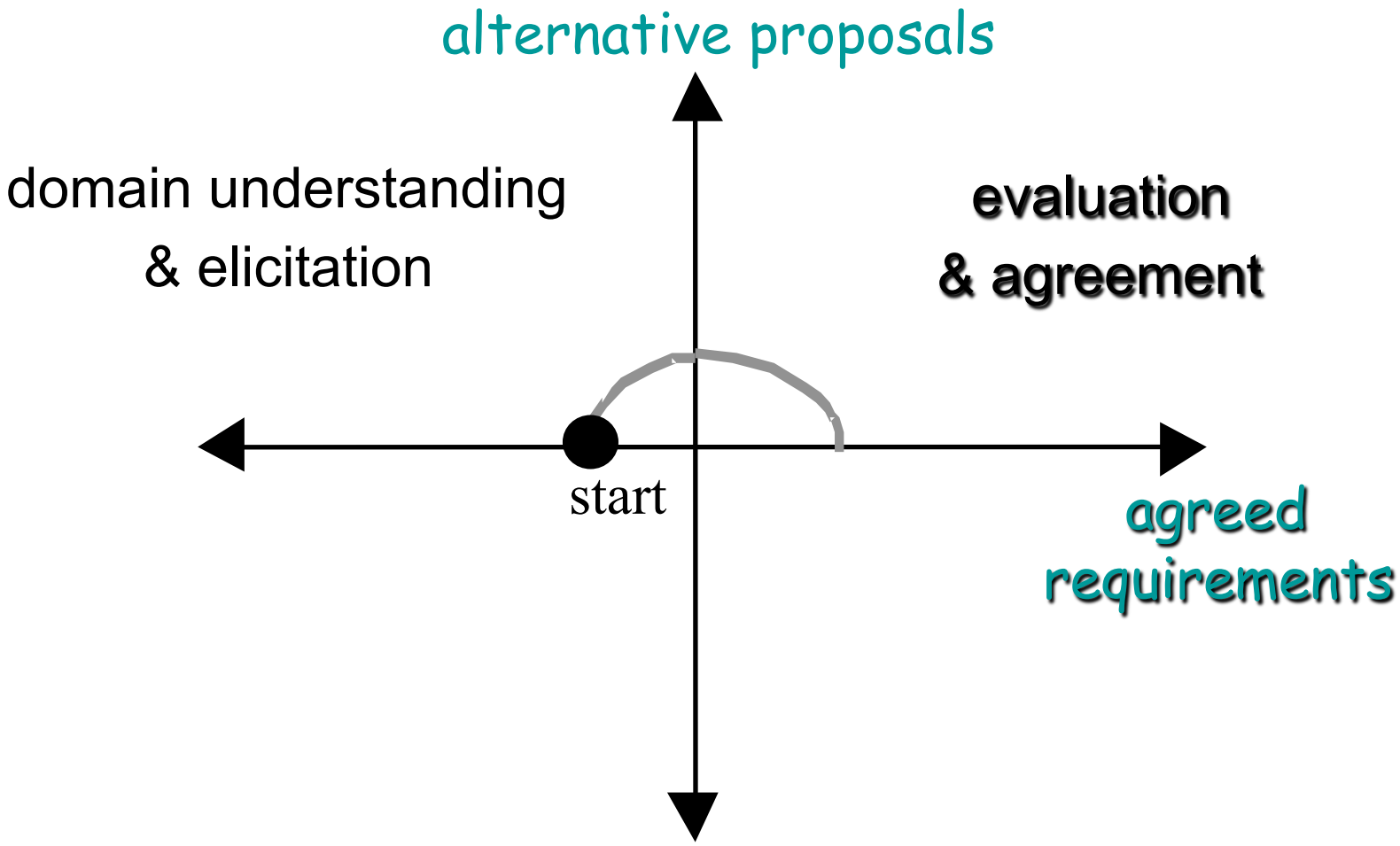
Requirements elicitation

Exploring the problem world ...

- Further analysis of problems with system-as-is
  - symptoms, causes, consequences
- Identification of
  - technology opportunities, market conditions
  - improvement objectives
  - organizational/technical constraints on system-to-be
  - *alternative* options for satisfying objectives, for assigning responsibilities
  - scenarios of hypothetical software-environment interaction
  - requirements on software, assumptions on environment

Result: Additional sections for preliminary draft proposal

The RE process

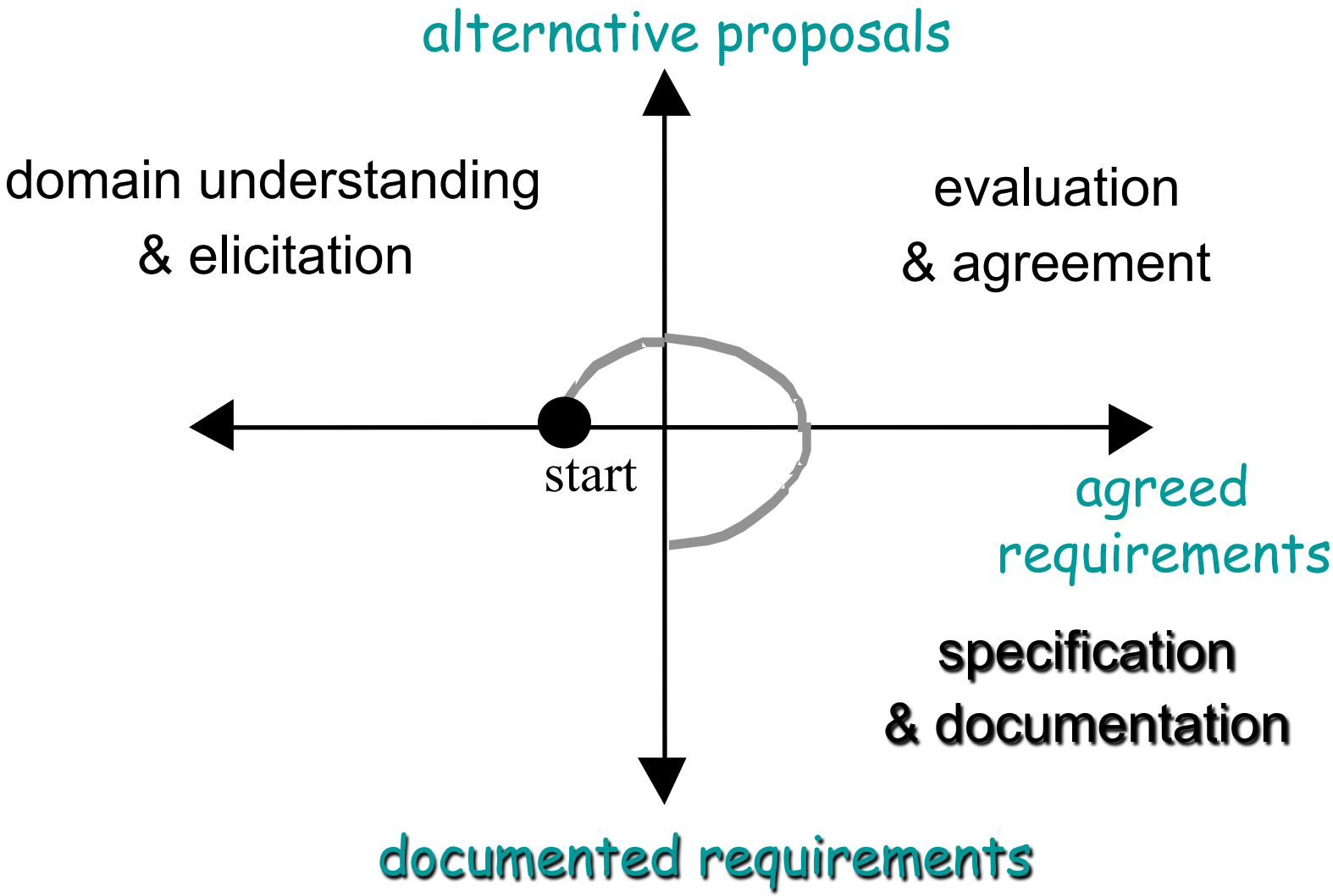


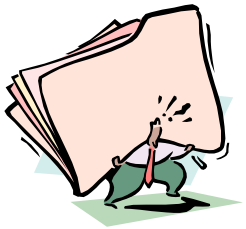
Evaluation & agreement

- Negotiation-based decision making
  - Identification & resolution of **conflicting** concerns
  - Identification & resolution of **risks** with proposed system
  - Comparison of **alternative options** against objectives & risks, and selection of preferred ones
  - Requirements **prioritization**: to resolve conflicts, address cost/ schedule constraints, support incremental development

Result: Final sections of draft proposal documenting the selected/agreed objectives, requirements, asssumptions (incl. rationale for the selected options)

The RE process



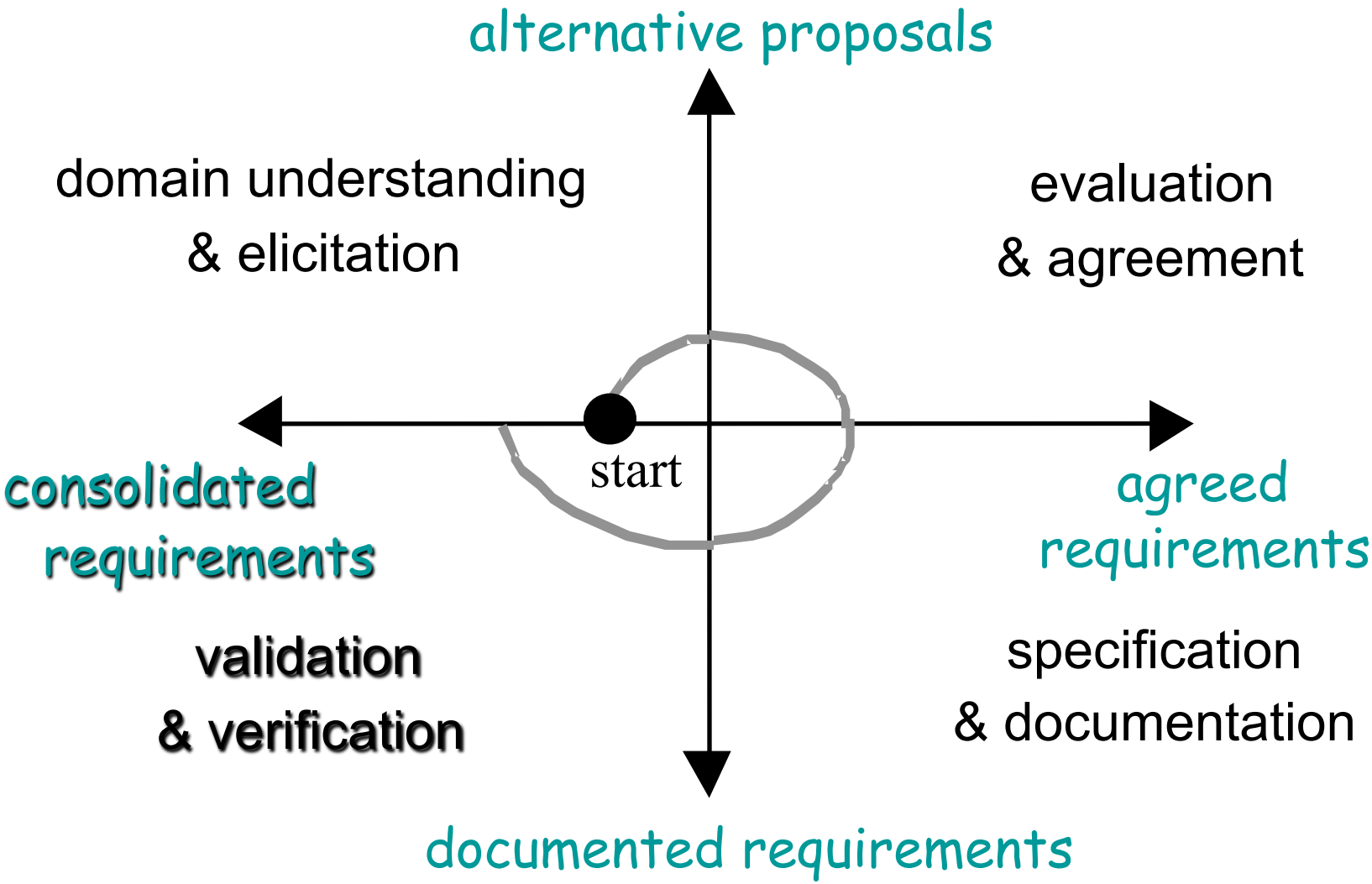


Specification & documentation

- Precise definition of all features of the agreed system
  - Objectives, concepts, relevant domain properties, system/ software requirements, assumptions, responsibilities
  - Likely system variants & evolutions
- Organization of these in a coherent structure
- Documentation in a form understandable by all parties

Result: **Requirements Document** (RD)

The RE process



Requirements consolidation

- Quality assurance activity on RD ...
  - Validation: adequacy of RD items wrt real needs?
  - Verification: omissions, inconsistencies?
  - Fixing of errors & flaws
- **Result**: Consolidated RD

RE: an iterative process

- Iterated cycles due to error corrections & **evolving needs**
  - during RE, during software development, after deployment