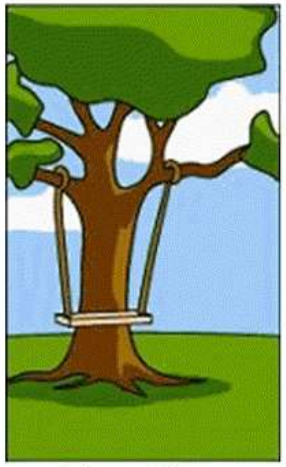
Foundations of Software Engineering

Lecture 5: Requirements are hard

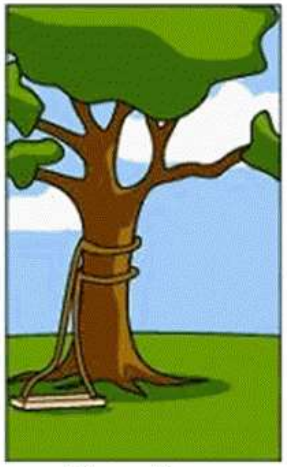
Christian Kästner

How the customer explained it



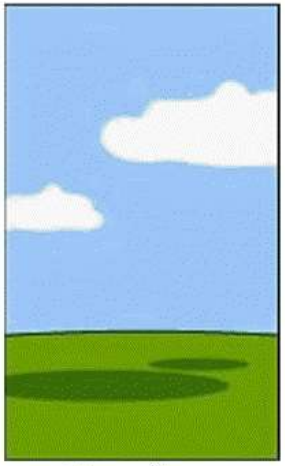
How the project leader understood it

How the engineer designed it

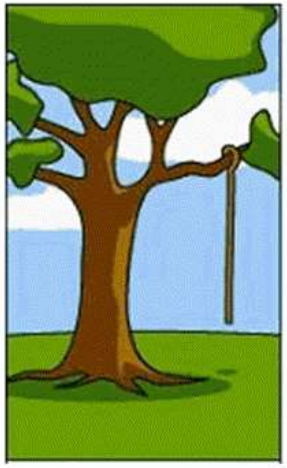


How the programmer wrote it

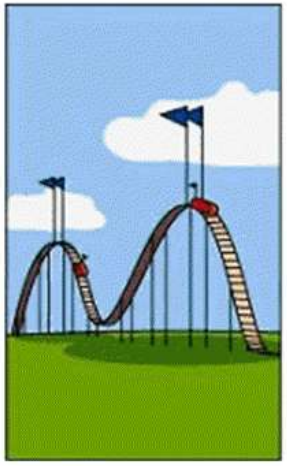
How the sales executive described it



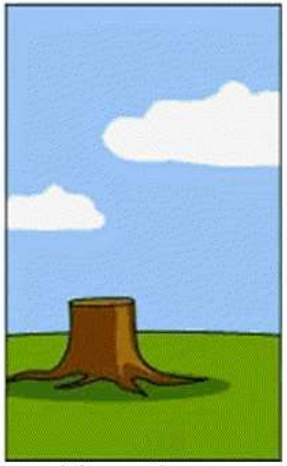
How the project was documented



What operations installed



How the customer was billed



How the help desk supported it

What the customer really needed

Learning goals

- Explain the importance and challenges of requirements in software engineering.
- Explain how and why requirements articulate the relationship between a desired system and its environment. Identify assumptions.
- Distinguish between and give examples of: functional and quality requirements; informal statements and verifiable requirements.
- State quality requirements in measurable ways

Overly simplified definition.

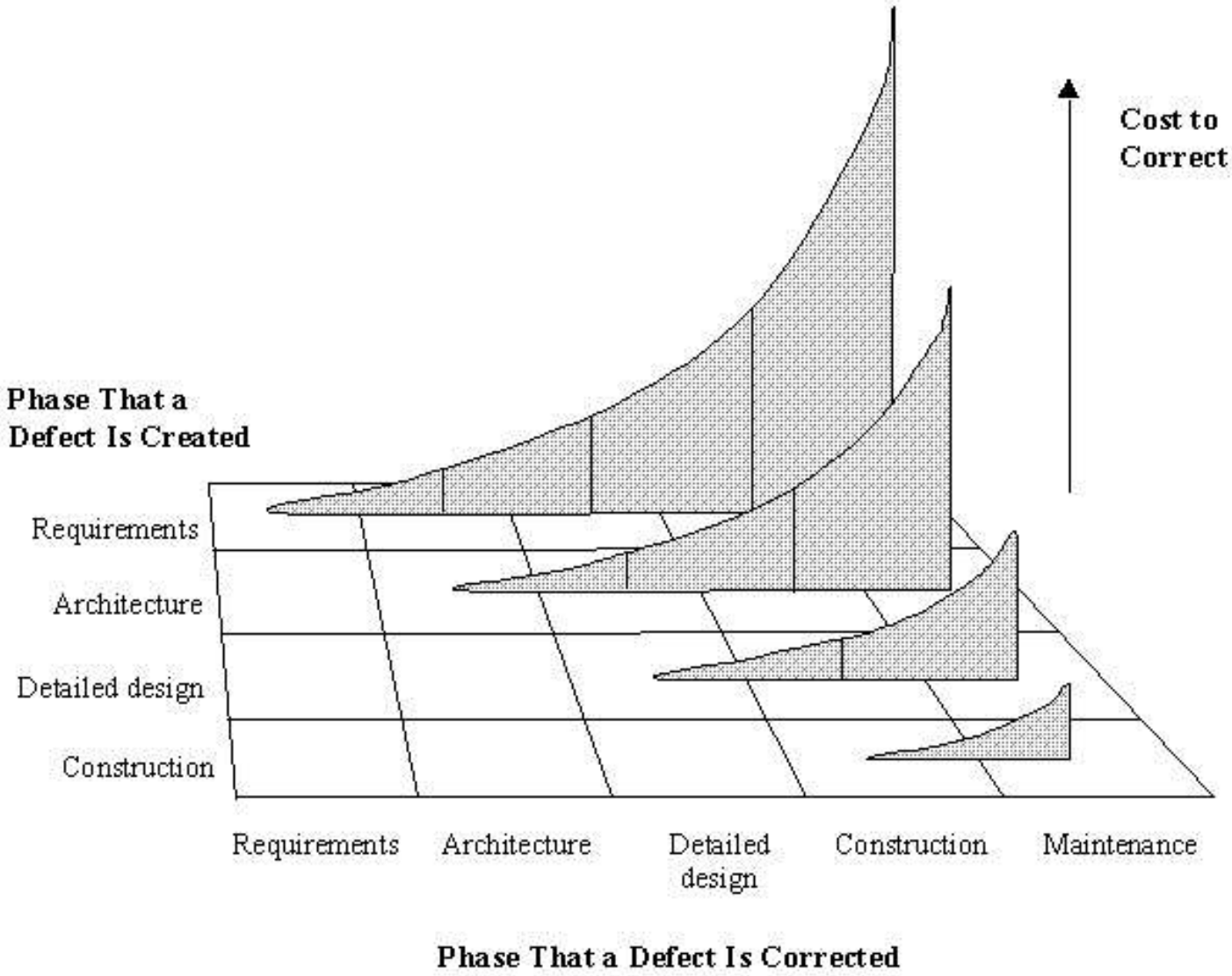
Requirements say what the system will do (and not how it will do it).

Healthcare.gov

Fred Brooks, on requirements.

- *The hardest single part of building a software system is deciding precisely **what to build**.*
- *No other part of the conceptual work is as difficult as establishing the detailed technical requirements ...*
- *No other part of the work so cripples the resulting system if done wrong.*
- *No other part is as difficult to rectify later.*

— Fred Brooks



A problem that stands the test of time…

A 1994 survey of 8000 projects at 350 companies found: 31% of projects canceled before completed; 9% of projects delivered on time, within budget in large companies, 16% in small companies.

- Similar results reported since.

Causes:

1. Incomplete requirements (13.1%)
2. Lack of user involvement (12.4%)
3. Lack of resources (10.6%)
4. Unrealistic expectations (9.9%)
5. Lack of executive support (9.3%)
6. Changing requirements and specifications (8.7%)
7. Lack of planning (8.1%)
8. System no longer needed (7.5%) .

Communication problem

Goal: figure out what should be built.

Express those ideas so that the correct thing is built.

EXAMPLE

Four Kinds of Denial

- Denial by prior knowledge – we have done this before, so we know **what** is required
- Denial by hacking – our fascination with machines dominates our focus on the **how**
- Denial by abstraction – we pursue elegant models which obscure, remove or downplay the real world
- Denial by vagueness – imply (vaguely) that machine descriptions are actually those of the world

Michael Jackson, "The World and the Machine," *International Conference on Software Engineering*, pp. 283-292, 1995.

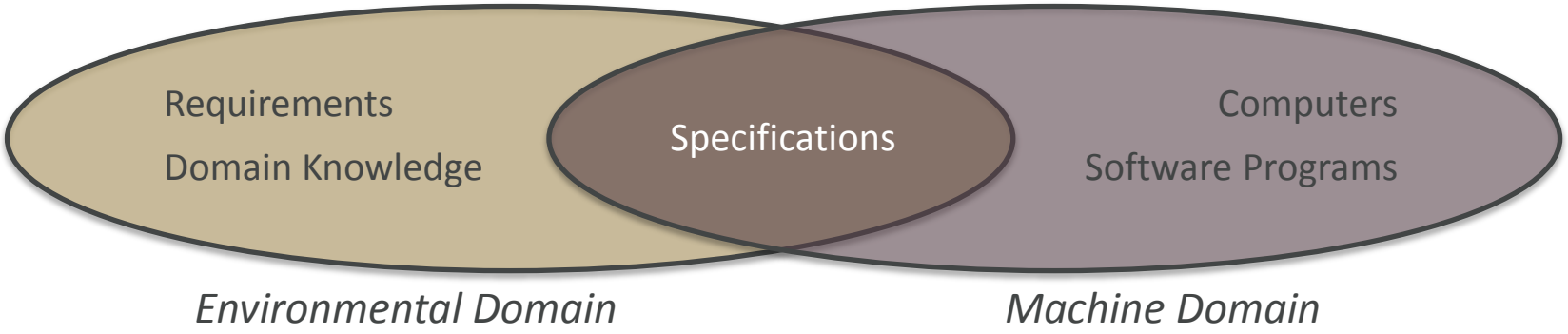
EXAMPLE 2, YOUR TURN

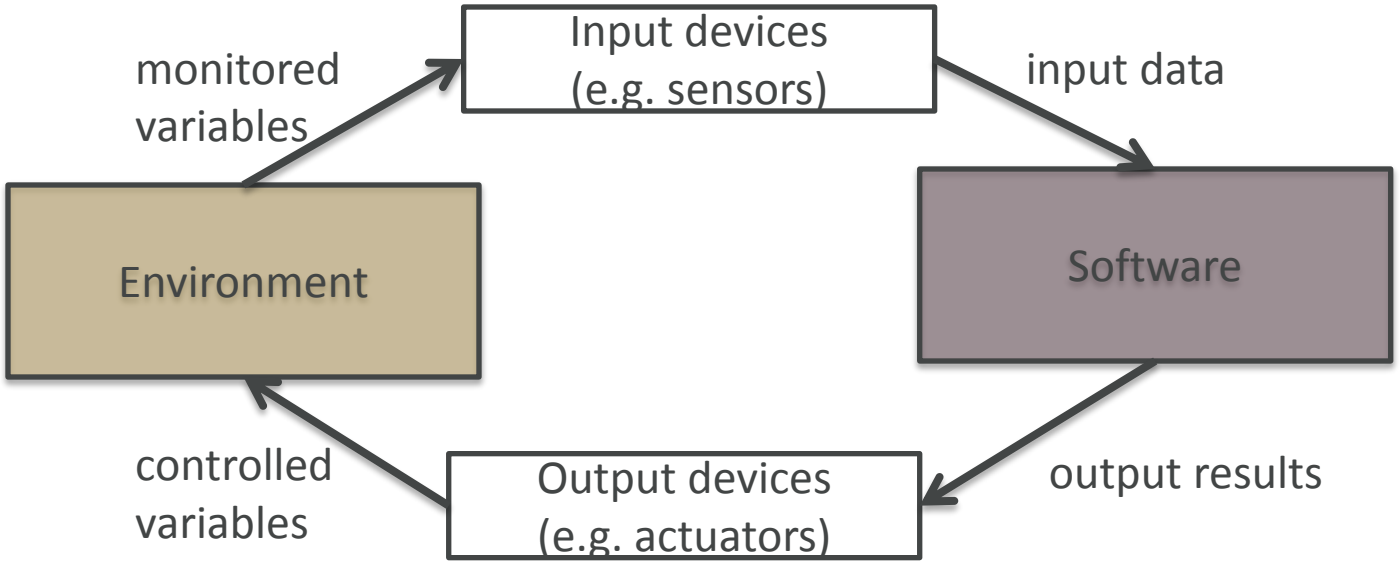
“Selling videos on the web”

- Involved subproblems?
- Required functionality?
- Nice to have functionality?
- Expected qualities?
- How fast to deliver at what quality for what price?

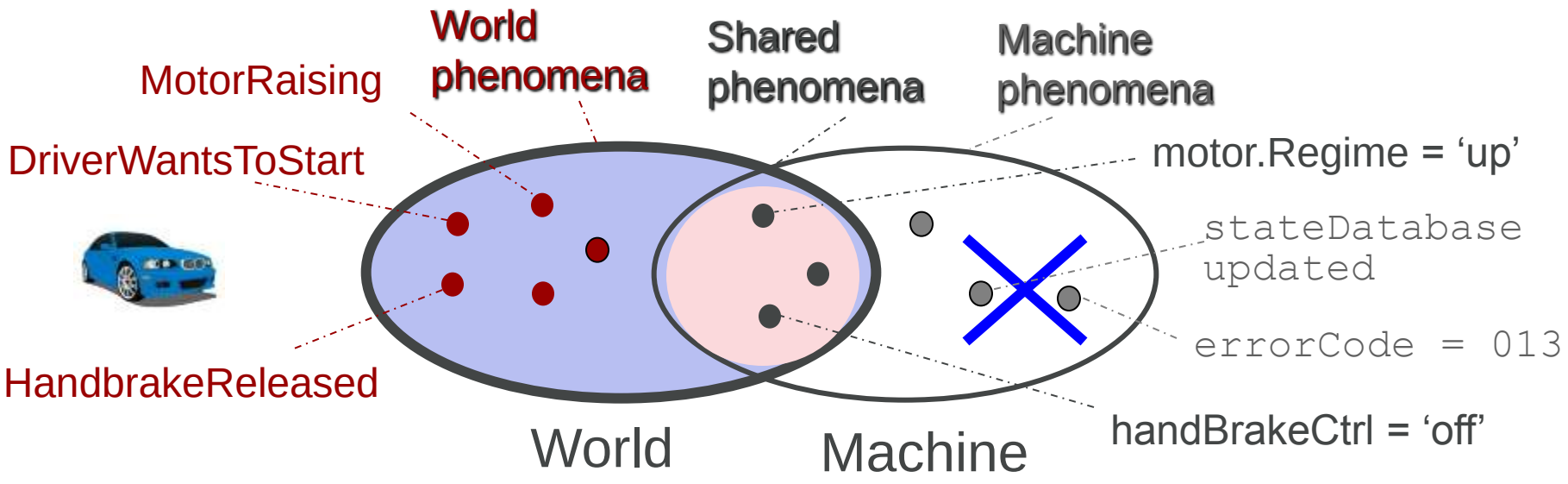
THE WORLD AND THE MACHINE

Environment and the Machine





Pamela Zave & Michael Jackson, "Four Dark Corners of Requirements Engineering," *ACM Transactions on Software Engineering and Methodology,* 6(1): 1-30, 1997.



Airbus Breaking System

- The Airbus A320-200 airplane has a software-based braking system that consists of:
  - Ground spoilers (wing plates extended to reduce lift)
  - Reverse thrusters
  - Wheel brakes on the main landing gear
- To engage the braking system, the **wheels of the plane must be on the ground**.

Is this a shared or an unshared action/condition?

Airbus Breaking System: System vs Software Requirements

- System requirements: relationships between monitored and controlled variables
- Software requirements: relationship between inputs and outputs
- Domain properties and assumptions state relationships between those

Sys.-Req., Soft.-Req., Assumptions for Airbus Breaking System

Lufthansa Flight 2904

Lufthansa Flight 2904

There are two "on ground" conditions:

1. Either shock absorber bears a load of 6300 kgs
2. Both wheels turn at 72 knots (83 mph) or faster

- Ground spoilers activate for conditions 1 or 2
- Reverse thrust activates for condition 1 on both main landing gears
- Wheel brake activation depends upon the rotation gain and condition 2

Actions of an ATM customer:

withdrawal-request(a, m)

Properties of the environment:

balance(b, p)

Actions of an ATM machine:

withdrawal-payout(a, m)

Properties of the machine:

expected-balance(b, p)

What other **models of the world** do machines maintain?

Online Shopping Example

- Stories: Scenarios and Use Cases

  "After the customer submits the purchase information and the payment has been received, the order is fulfilled and shipped to the customer's shipping address."

- Optative statements

  The system *shall* notify clients about their shipping status

- Domain Properties and Assumptions

  Every product has a unique product code

  Payments will be received after authorization

IMPLEMENTATION BIAS

Requirements say what the system will do (**and not how it will do it**).

Why not “how”?

Avoiding implementation bias

- Requirements describe what is observable at the environment-machine interface.
- *Indicative mood* describes the environment (as-is)
- *Optative mood* to describe the environment with the machine (to-be).

QUALITY REQUIREMENTS

Functional Requirements

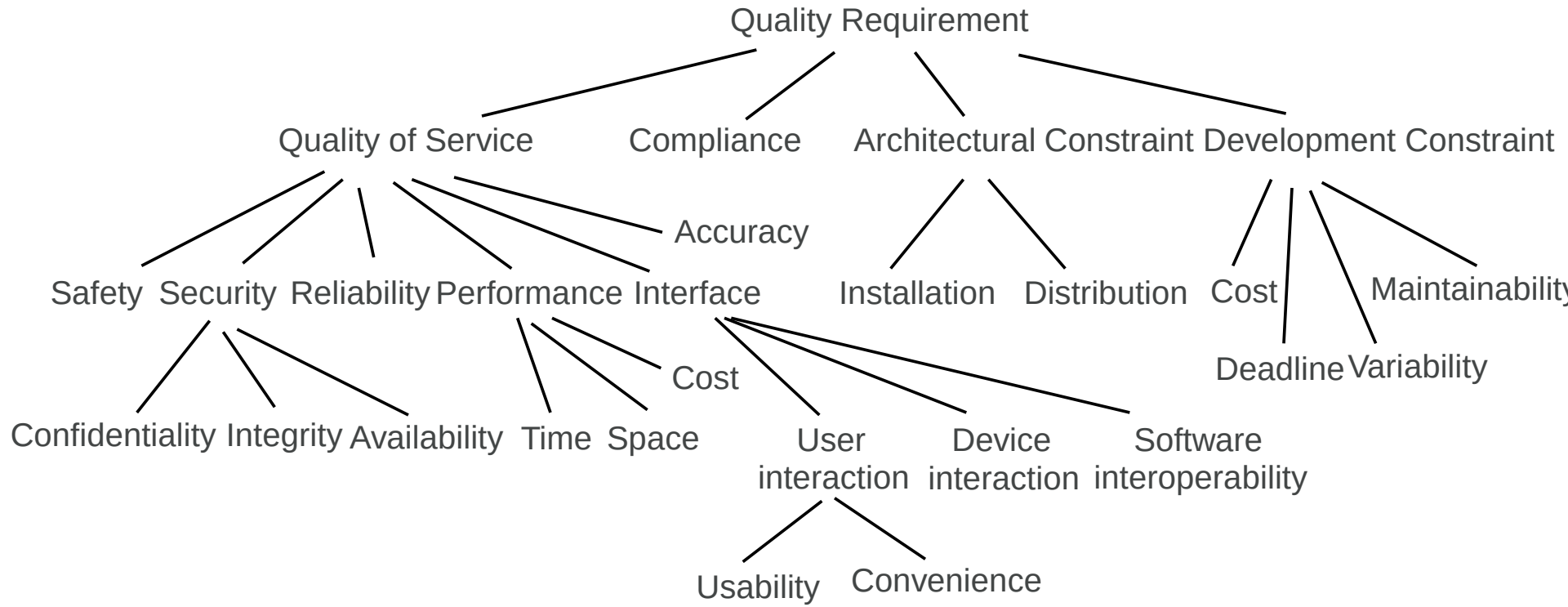
- What the machine should do
  - Input
  - Output
  - Interface
  - Response to events
- Criteria
  - Completeness: All requirements are documented
  - Consistency: No conflicts between requirements
  - Precision: No ambiguity in requirements

Quality (non-funct.) requirements

- Specify not the functionality of the system, but the quality with which it delivers that functionality.
- Can be more critical than functional requirements
  - Can work around missing functionality
  - Low-quality system may be unusable
- Examples?

Here's the thing…

- Who is going to ask for a slow, inefficient, unmaintainable system?
- A better way to think about quality requirements is as *design criteria to help choose between alternative implementations.*
- Question becomes: to what extent must a product satisfy these requirements to be acceptable?



Selling videos on the web?

Expressing quality requirements

- Requirements serve as contracts: they should be testable/falsifiable.
- Informal goal: a general intention, such as ease of use.
  - May still be helpful to developers as they convey the intentions of the system users.
- Verifiable non-functional requirement: A statement using some measure that can be objectively tested.

Requirements metrics

Property	Measure

Examples

- **Informal goal:** “the system should be easy to use by experienced controllers, and should be organized such that user errors are minimized.”
- **Verifiable non-functional requirement:** “Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day, on average.”

Examples

- Confidentiality requirement: A non-staff patron may never know which books have been borrowed by others.
- Privacy requirement: The diary constraints of a participant may never be disclosed to other invited participants without his or her consent.
- Integrity req: The return of book copies shall be encoded correctly and by library staff only.
- Availability req: A blacklist of bad patrons shall be made available at any time to library staff. Information about train positions shall be available at any time to the vital station computer.

Source: van Lamsweerde. Requirements Engineering: … Wiley 2009

Examples 2

- Reliability req: The train acceleration control software shall have a mean time between failures of the order of 109 hours.
- Accuracy req: A copy of a book shall be stated as available by the loan software if and only if it is actually available on the library shelves. The information about train positions used by the train controller shall accurately reflect the actual position of trains up to X meters at most. The constraints used by the meeting scheduler should accurately reflect the real constraints of invited participants.
- Performance req: Responses to bibliographical queries shall take less than 2 seconds. Acceleration commands shall be issued to every train every 3 seconds. The meeting scheduler shall be able to accommodate up to x requests in parallel. The new e-subscription facility should ensure a 30% cost saving.

Examples 3

- Interface req: The format for bibliographical queries and answers shall be accessible to students from any department. To ensure smooth and comfortable train moves, the difference between the accelerations in two successive commands sent to a train should be at most x. To avoid disturbing busy people unduly, the amount of interaction with invited participants for organizing meetings should be kept as low as possible.
- Interoperability req: The meeting scheduling software should be interoperable with the wss Agenda Manager product.
- Compliance req: The value for the worst-case stopping distance between successive trains shall be compliant with international railways regulations. The meeting scheduler shall by default exclude official holidays associated with the target market.

Examples 4

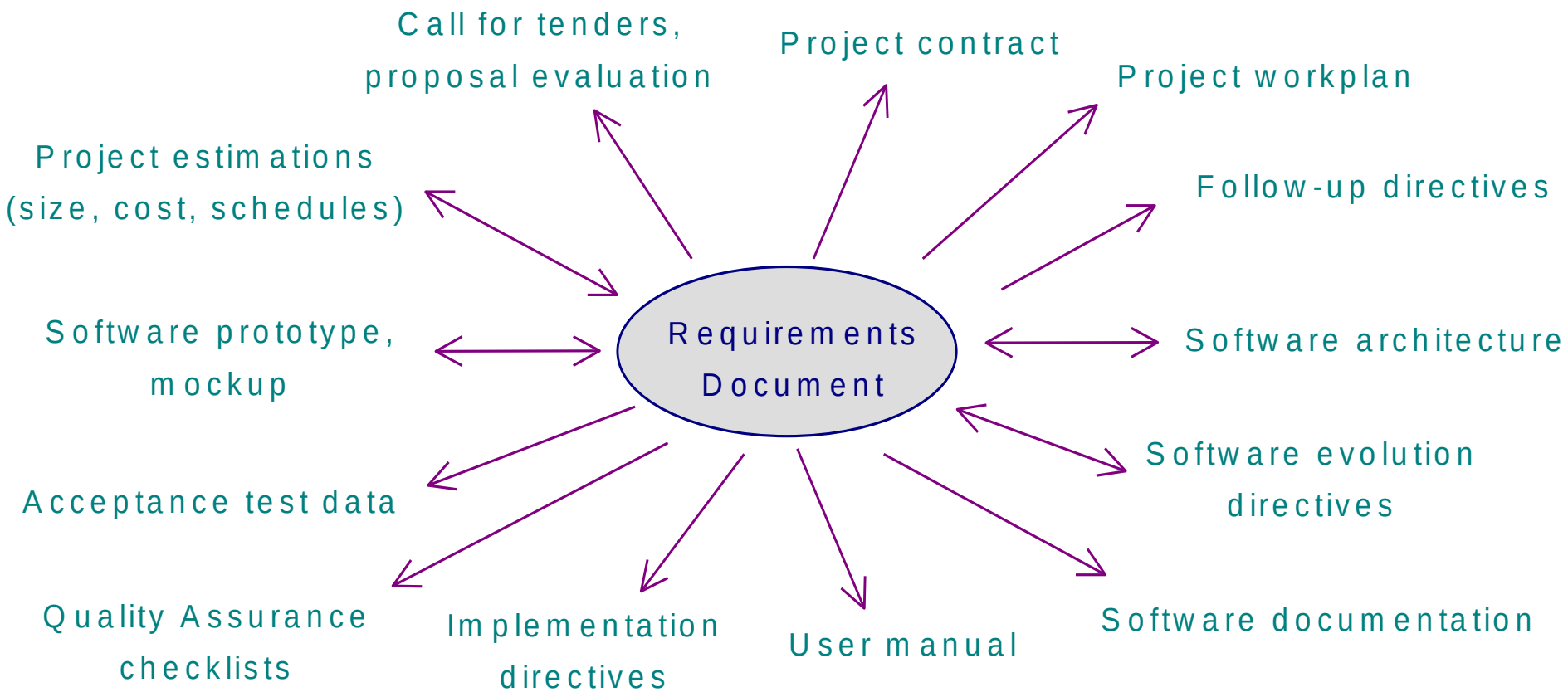
- Architectural req: The on-board train controllers shall handle the reception and proper execution of acceleration commands sent by the station computer. The meeting scheduling software should run on Windows version X.x and Linux version Y.y.
- Development req.: The overall cost of the new UWON library software should not exceed x. The train control software should be operational within two years. The software should provide customized solutions according to variations in type of meeting (professional or private, regular or occasional), type of meeting location (fixed, variable) and type of participant (same or different degrees of importance).

ACTIVITIES OF REQUIREMENTS ENGINEERING

What is requirements engineering?

- Knowledge **acquisition** – how to capture relevant detail about a system?
  - Is the knowledge complete and consistent?
- Knowledge **representation** – once captured, how do we express it most effectively?
  - Express it for whom?
  - Is it received consistently by different people?
- You may sometimes see a distinction between the requirements *definition* and the requirements *specification*.

Requirements in software projects



Why, What, Who of RE

System-as-is

problems,
opportunities,
system knowledge

System-to-be

Objectives

satisfy

requirements,
constraints,
assumptions

WHY
a new system?

WHAT
services?

assignment

WHO
will be
responsible
for what ?

Typical Steps (Iterative)

- Identifying stakeholders
- Domain understanding
- Requirements elicitation (interviews, …)
- Evaluation and agreement (conflicts, priorization, risks, …)
- Documentation/specification
- Consolidation / quality assurance

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

Types of RE errors & flaws

- Omission (critical error!)
- Contradiction (critical error!)
- Inadequacy (critical error!)
- Ambiguity (critical error!)
- Unmeasurability
- Noise, overspecification
- Unfeasibility (wishful thinking)
- Unintelligibility
- Poor structuring, forward reference, remorse
- Opacity

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
  - "Panels inside trains shall display all flights served at next stop"
- Ambiguity: RD item allowing a problem world feature to be interpreted in different ways
  - "Doors shall be open as soon as the train is stopped at platform"
- 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 (RD)

- 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
  - A requirement statement containing 5 acronyms
- 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 doc. (2)

- 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

Documenting requirements

- Free unrestricted text
- Structured text
- Diagrams
- Formal specifications

Further Reading

- Van Lamsweerde A. Requirements engineering: From system goals to UML models to software. John Wiley & Sons; 2009. Chapter 1