ECE444: Software Engineering

Requirements 1: Overview and Concepts

Shurui Zhou

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Administrivia

Lab 1: Git&GitHub

4 activities, submit your repo url by Friday

(command line / desktop UI)

Vote for ideas: think about feasibility on collecting requirement

Milestone 1: Team workflow

Lab2-Lab5 Flask

Learning Goals for last lecture (Intro of Process)

- Recognize the Importance of process
- Understand the difficulty of measuring progress
- Use milestones for planning and progress measurement
- Understand backlogs and user stories

Learning Goals

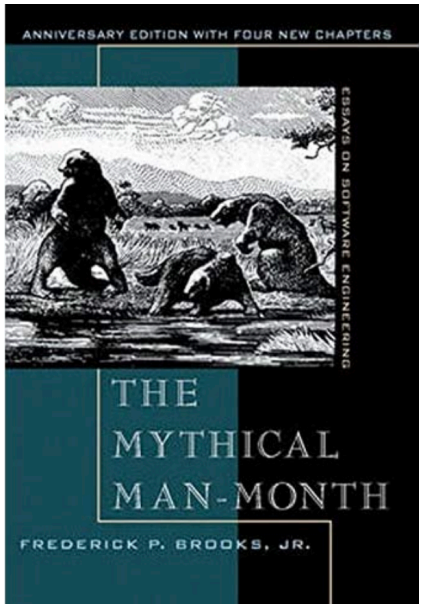
- Explain the importance and challenges of requirements in software engineering.
- Explain how and why requirements articulate the relationship between a desired system and its environment.
- Identify assumptions.
- Distinguish between and give examples of: functional and quality requirements; informal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

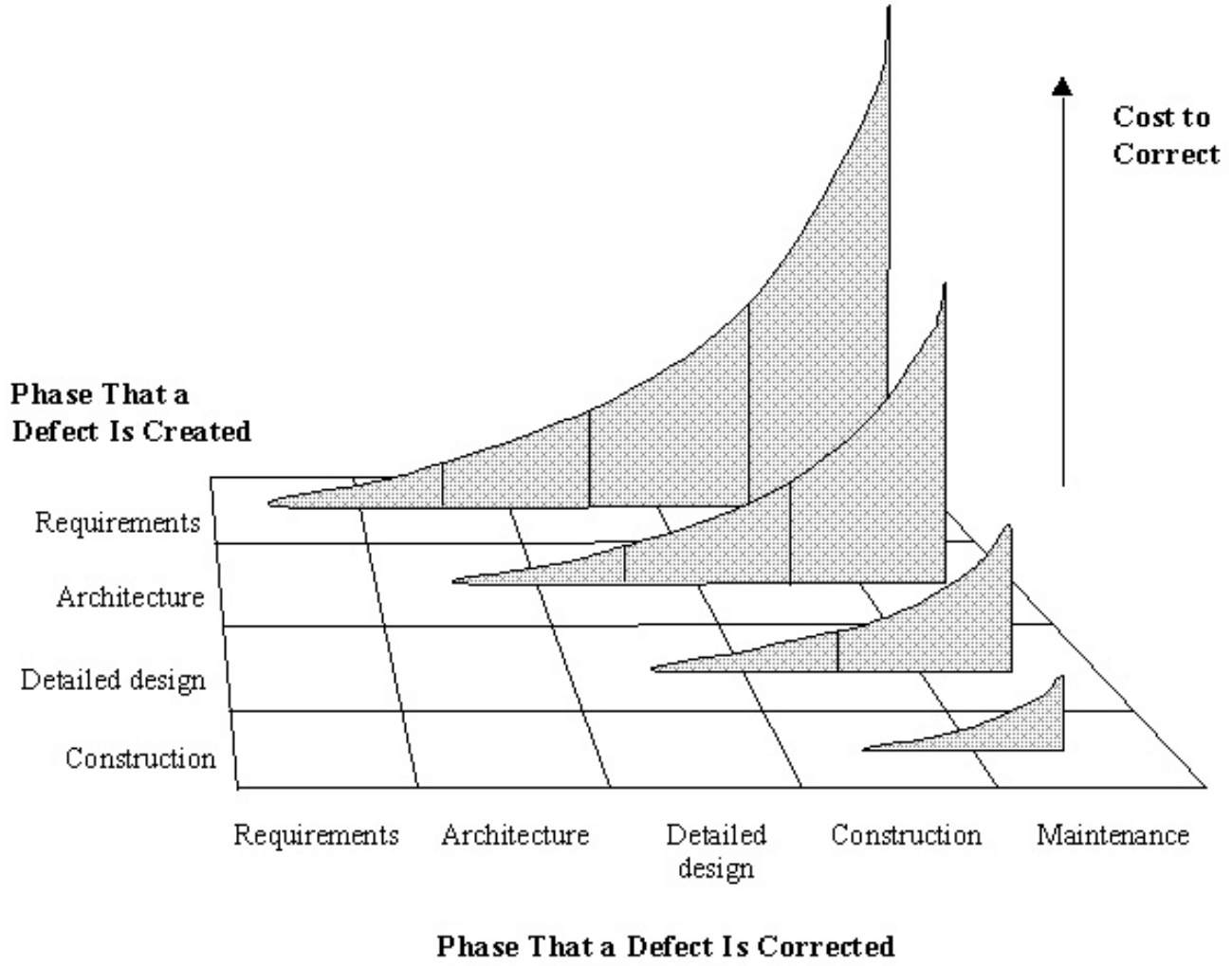
Overly simplified definition

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

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 is as difficult to rectify later.*





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%) .

The Standish Group Report

CHAOS

"The Roman bridges of antiquity were very inefficient structures. By modern standards, they used too much stone, and as a result, far too much labour to build. Over the years we have learned to build bridges more efficiently, using fewer materials and less labour to perform the same task."

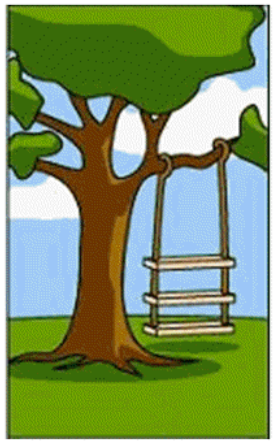
-Tom Clancy (The Sum of All Fears)

Why is this hard?

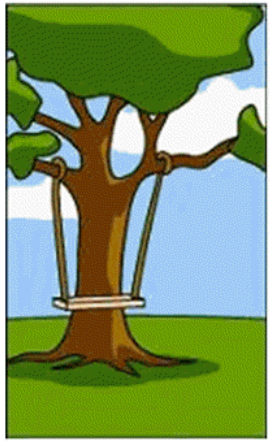
Communication problem

Goal: figure out what should be built.

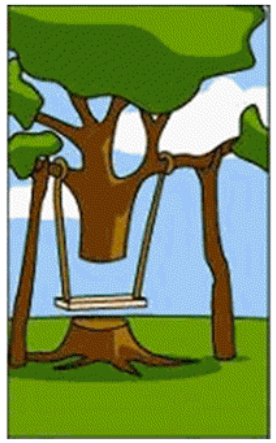
Express those ideas so that the correct thing is built.



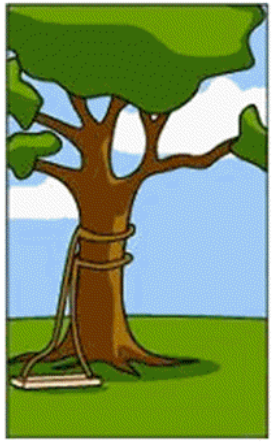
How the customer explained it



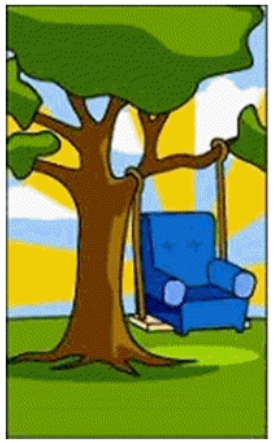
How the project leader understood it



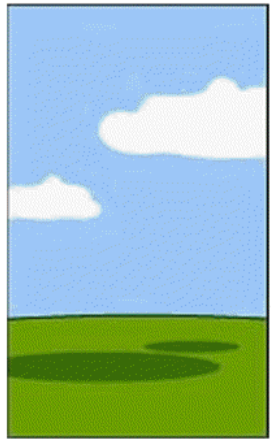
How the engineer designed it



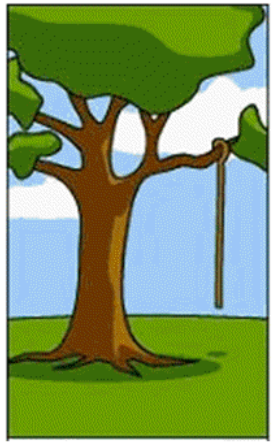
How the programmer wrote it



How the sales executive described it

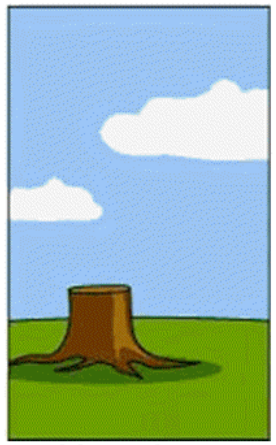


How the project was documented

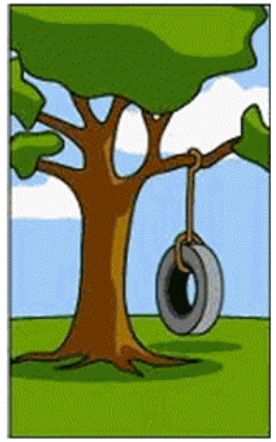


What operations installed

How the customer was billed



How the help desk supported it



What the customer really needed

What is requirement engineering?

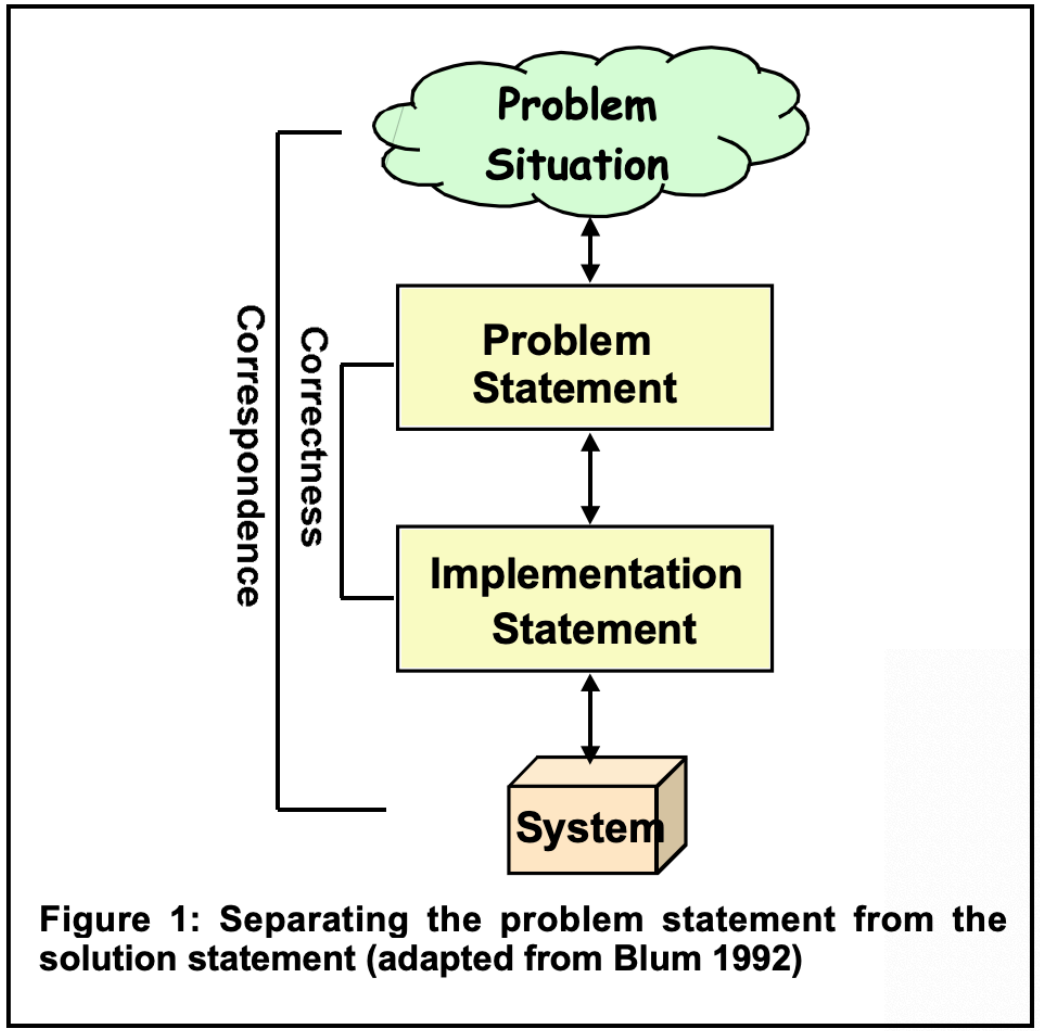


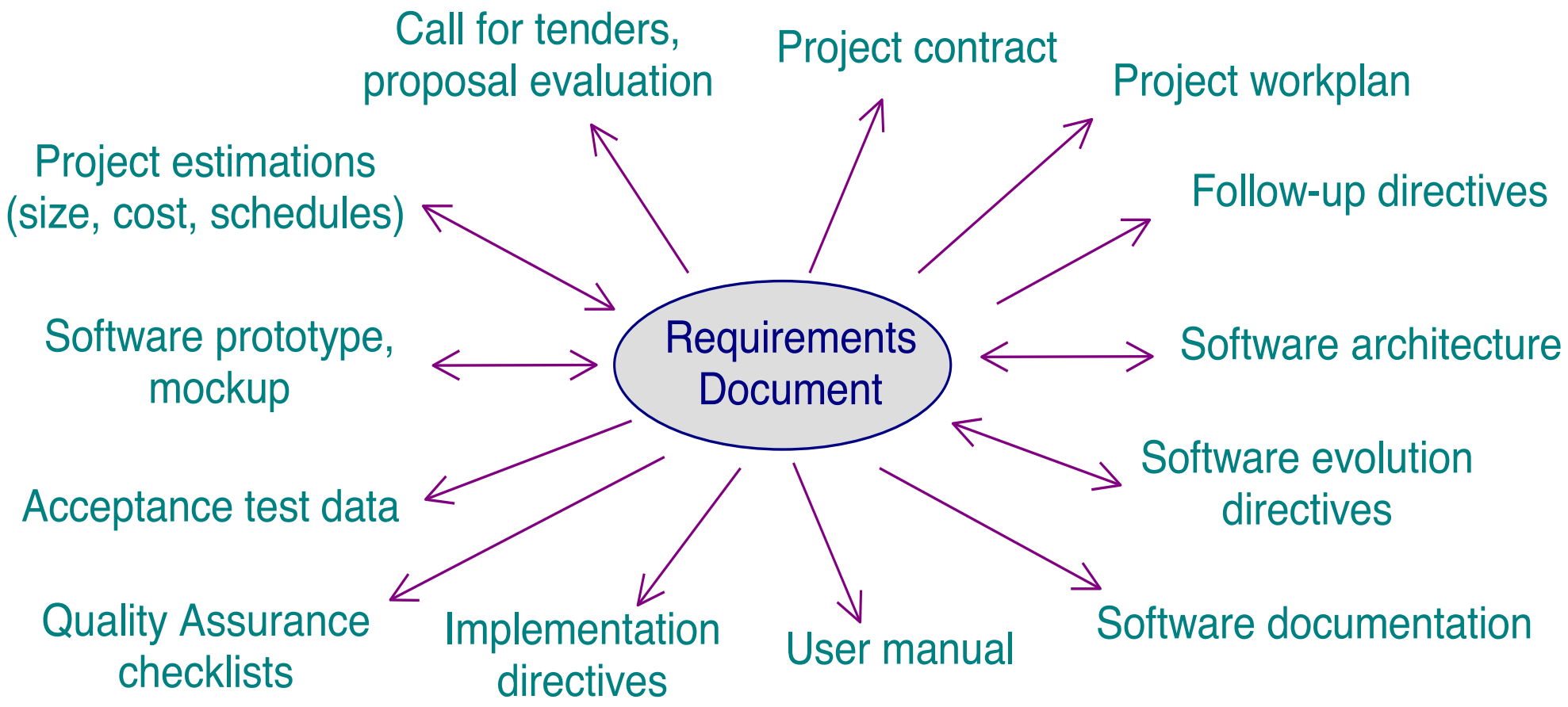
Figure 1: Separating the problem statement from the solution statement (adapted from Blum 1992)

- http://www.cs.toronto.edu/~sme/CSC340F/readings/FoRE-chapter02-v7.pdf

What is requirement 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?

Requirements in software projects



User and System Requirements

User Requirements

- It describes the services that the system should provide and the constrains under which it must operate.
- We don’t expect to see any level of detail, or what exactly the system will do, It’s more of generic requirements.
- It’s usually written in a natural language and supplied by diagrams.

System Requirements

- a more detailed description of the system services and the operational constrains such as how the system will be used, and development constrains such as the programming languages.
- audiences: engineers, system architects, testers, etc.

Less simplified definition – Online Shopping

- 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

Capturing vs Synthesizing

- Engineers acquire requirements from many sources
  - Elicit from stakeholders
  - Extract from policies or other documentation
  - Synthesize from above + estimation and invention
- Because stakeholders do not always “know what they want”*, engineers must…
  - Be faithful to stakeholder needs and expectations
  - Anticipate additional needs and risks
  - Validate that “additional needs” are necessary or desired

Functional & Non-Functional Requirements

- **Functional Requirements**

It covers the main functions that should be provided by the system.

- *user* requirement, they are usually descried in an abstract way.

- *system* requirement describe the system functions, it's inputs, processing; how it's going to react to a particular input, and what's the expected output.

- **Non-Functional Requirements**

These are the constrains on the functions provided by the system.

e.g., performance & security &…

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

Keyword	Meaning
MUST	This word, or the terms “REQUIRED” or “SHALL”, mean that the definition is an **absolute requirement** of the specification.
MUST NOT	This phrase, or the phrase “SHALL NOT”, mean that the definition is an **absolute prohibition** of the specification.
SHOULD	This word, or the adjective “RECOMMENDED”, mean that there **may** exist valid reasons in particular circumstances to **ignore** a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
SHOULD NOT	This phrase, or the phrase “NOT RECOMMENDED” mean that there **may exist** valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
MAY	This word, or the adjective “OPTIONAL”, mean that an item is truly **optional**. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

- https://www.ietf.org/rfc/rfc2119.txt

Quality/Non-functional 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

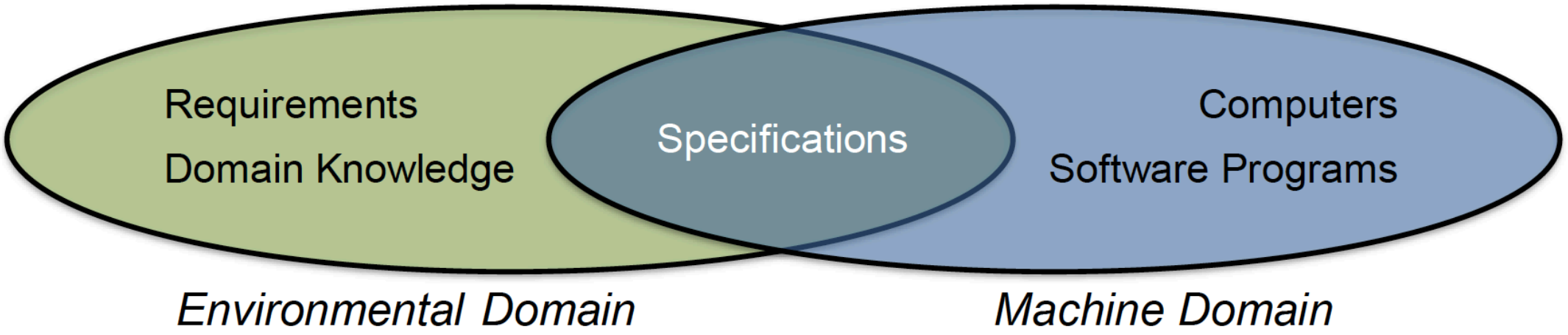
Functional requirements and implementation bias

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

Why not “how”?

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

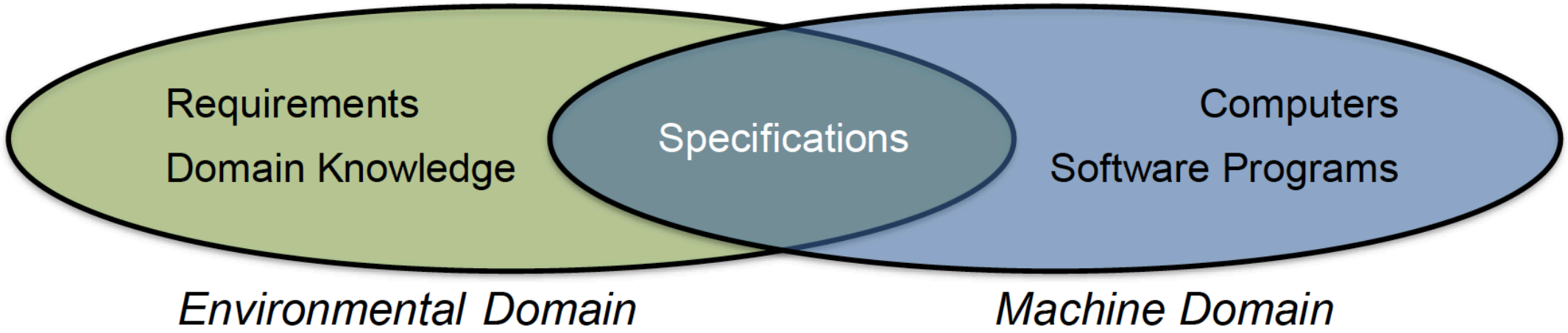
The World and the Machine



Three components

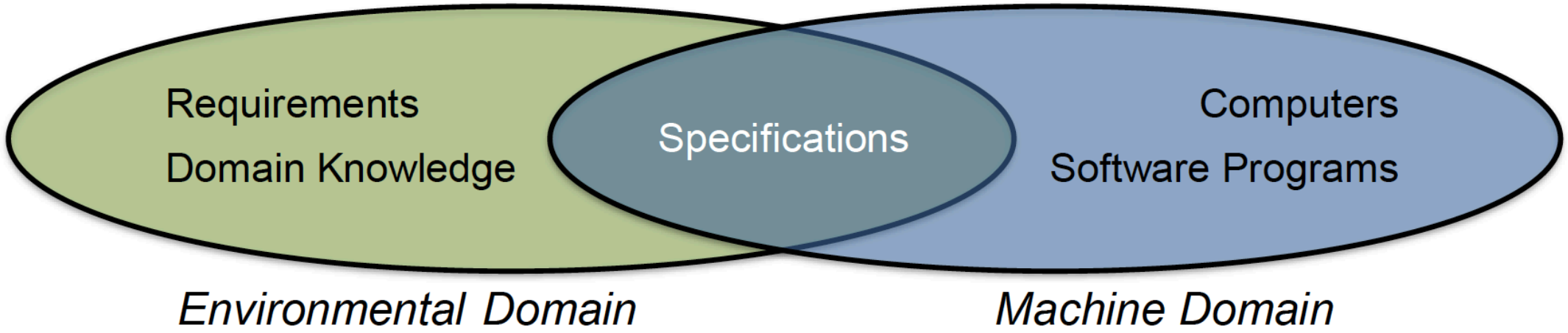
- **Requirements** (which are things in the world we would like to achieve)
- **Specifications** (which are descriptions of what the system we are designing should do if it is to meet the requirements)
- **Domain properties** (which are things that are true of the world anyway)

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



- The pilot shall decrease airspeed and lower the landing gear prior to decision height
- The plane shall lower the wing flaps to 30°for landing

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



Only a manager can assign access authority

Prevent access to unauthorized personnel

“only authorized personnel get access to a building ”

Shared- and unshared actions

- Actions are environment-or machine-controlled
- Actions either:
  - Shared with (belongs to, is observable by) the machine
  - Unshared, and not observable by the machine
- Actions in a Turnstile: shared or unshared?
  - pay
  - push
  - enter

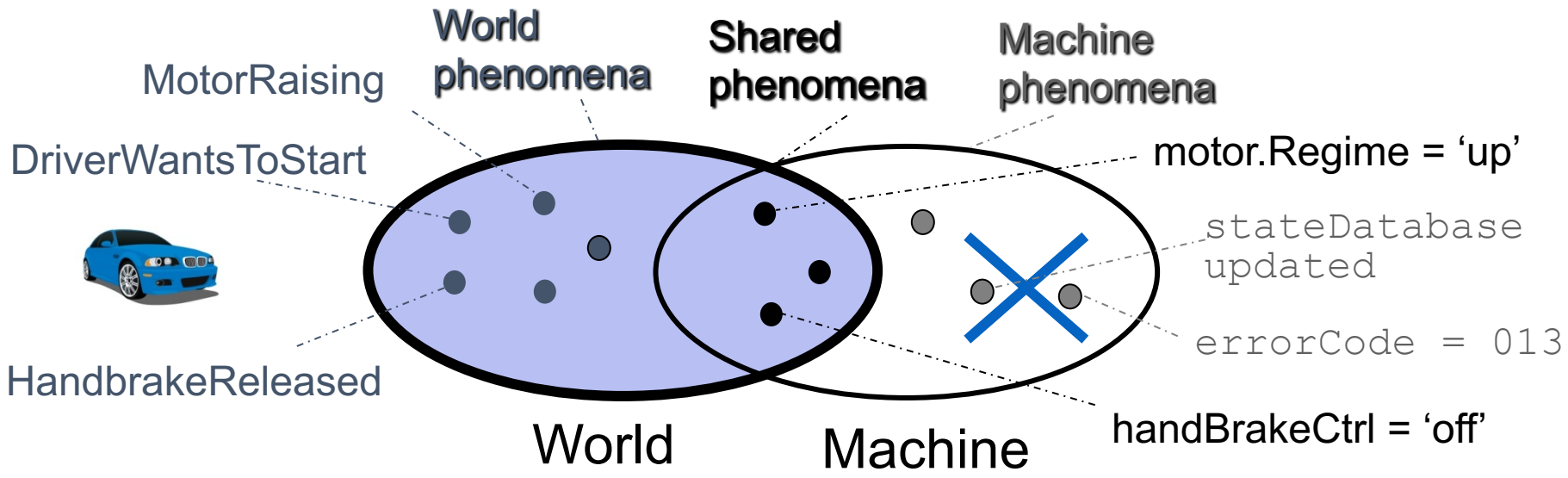
Shared- and unshared actions

- Actions are environment-or machine-controlled
- Actions either:
  - Shared with (belongs to, is observable by) the machine
  - Unshared, and not observable by the machine

Actions in a Turnstile

	Shared	Unshared
Actions	pay, push	enter

Shared- and unshared actions



Some gaps must remain…

- Unshared actions cannot be accurately expressed in the machine
  - People can jump over gates (enter without unlocking)
  - People can steal or misplace inventory

Three components

- **Requirements** (which are things in the world we would like to achieve)
- **Specifications** (which are descriptions of what the system we are designing should do if it is to meet the requirements)
- **Domain properties** (which are things that are true of the world anyway)

What are specifications?

- Only be written in terms of the shared phenomena between the machine domain and the environment
- Example: “only authorized personnel get access to a building ”
  - Machine space - prevent access to unauthorized personnel
  - World space - only a manager can assign access authority
  - specification - when the user enters a valid password, the computer will unlock the door

Domain properties

- help to link the specification and the requirements
- “only authorized personnel get access to a building ”
- Whether domain **properties** hold depends on the context:

  access control for an office environment

  vs

  a care home for elderly

Verification and Validation

Verification – is the software the correct?

- Does the software satisfy the specification?
- Is the specification correct with respect to the requirements, assuming the domain properties hold?

Validation – are the requirements correct?

- Are the requirements complete, or do the requirements accurately reflect the client's problem?
- Are the requirements consistent?

Verification and Validation

Verification – is the software the correct?

- Does the software satisfy the specification?
- Is the specification correct with respect to the requirements, assuming the domain properties hold?

Validation – are the requirements correct?

- Are the requirements complete, or do the requirements accurately reflect the client's problem?
- Are the requirements consistent?

Automotive Industry

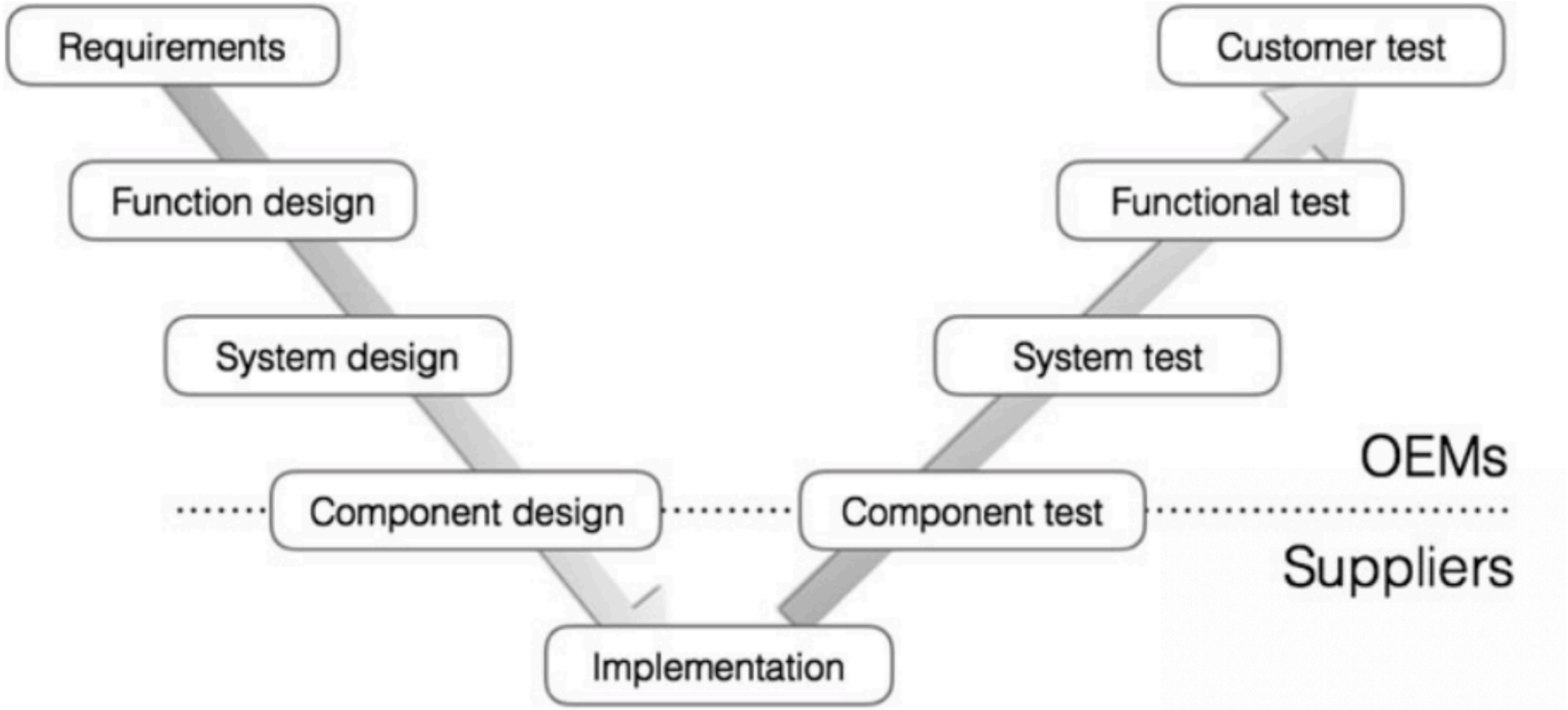


Fig. 3.1 V-shaped model of software development process in automotive software development

International industry standards for development of safety-critical systems

Airbus Braking 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."

→ **Req**: Reverse thrust should be enabled only when the aircraft is moving on the runway, and disabled at all other times

Airbus Braking System

"To engage the braking system, the wheels of the plane must be on the ground."

→ **Req**: Reverse thrust should be enabled only when the aircraft is moving on the runway, and disabled at all other times

→ **Spec**: reverse thrust should be enabled if and only if wheel pulses are on

2 Assumptions:

1. wheel pulses are on if and only if wheels are turning
2. wheels are turning if and only if aircraft is moving on the runway

Verification and Validation

Verification – is the software the correct?

- Does the software satisfy the specification?
- Is the specification correct with respect to the requirements, assuming the domain properties hold?

Validation – are the requirements correct?

- Are the requirements complete, or do the requirements accurately reflect the client's problem?
- Are the requirements consistent?

Airbus Braking System

"To engage the braking system, the wheels of the plane must be on the ground."

→ **Req**: Reverse thrust should be enabled only when the aircraft is moving on the runway, and disabled at all other times

→ **Spec**: reverse thrust should be enabled if and only if wheel pulses are on

2 Assumptions:

1. wheel pulses are on if and only if wheels are turning
2. wheels are turning if and only if aircraft is moving on the runway

Lufthansa Flight 2904 (1993)

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

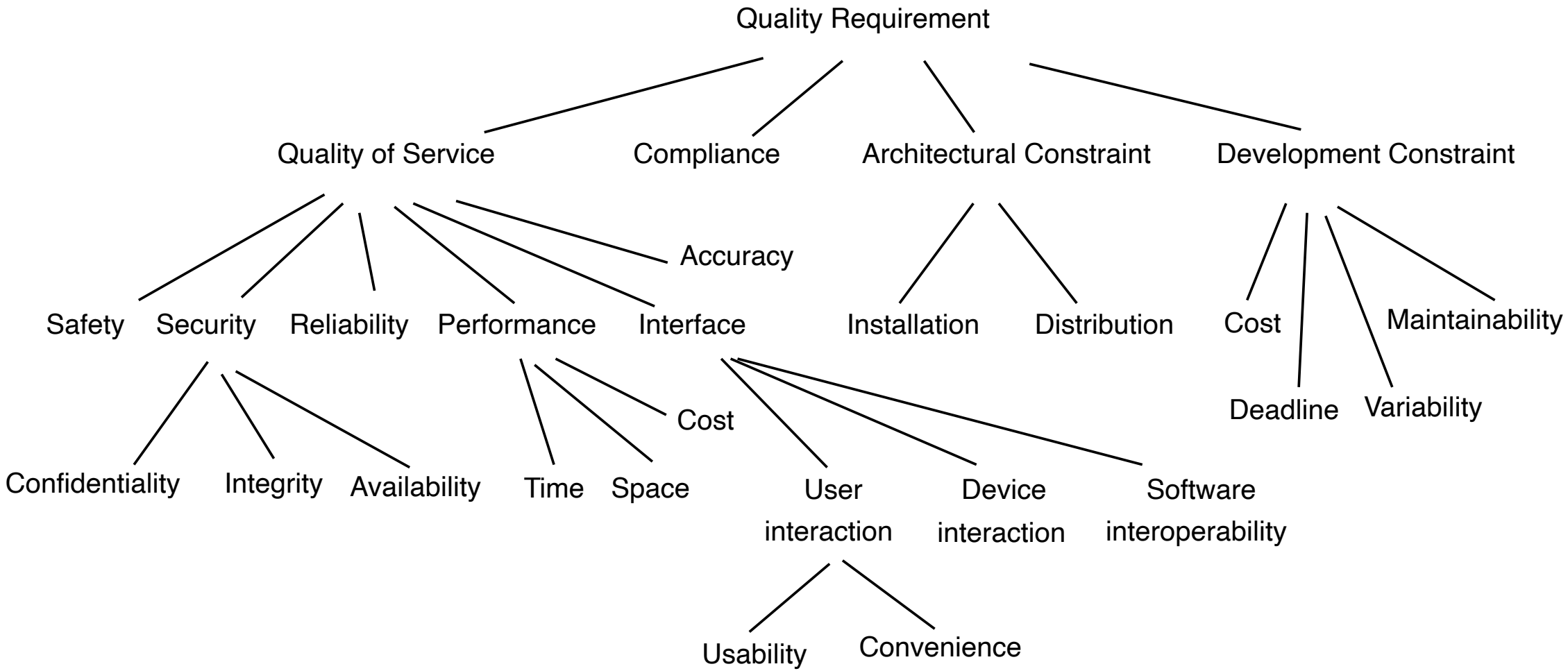
Quality Requirements

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?

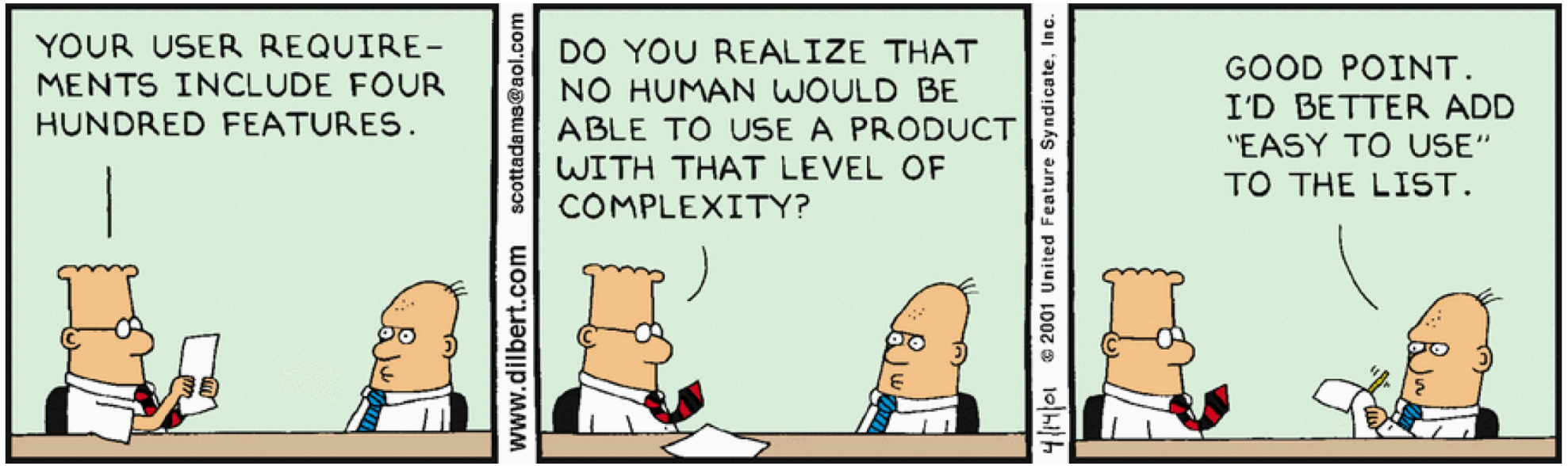
Here's the thing…

- 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?


Quality Requirement
Quality of Service
Compliance
Architectural Constraint
Development Constraint
Accuracy
Safety
Security
Reliability
Performance
Interface
Installation
Distribution
Cost
Maintainability
Deadline
Variability
Cost
Confidentiality
Integrity
Availability
Time
Space
User interaction
Device interaction
Software interoperability
Usability
Convenience

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.

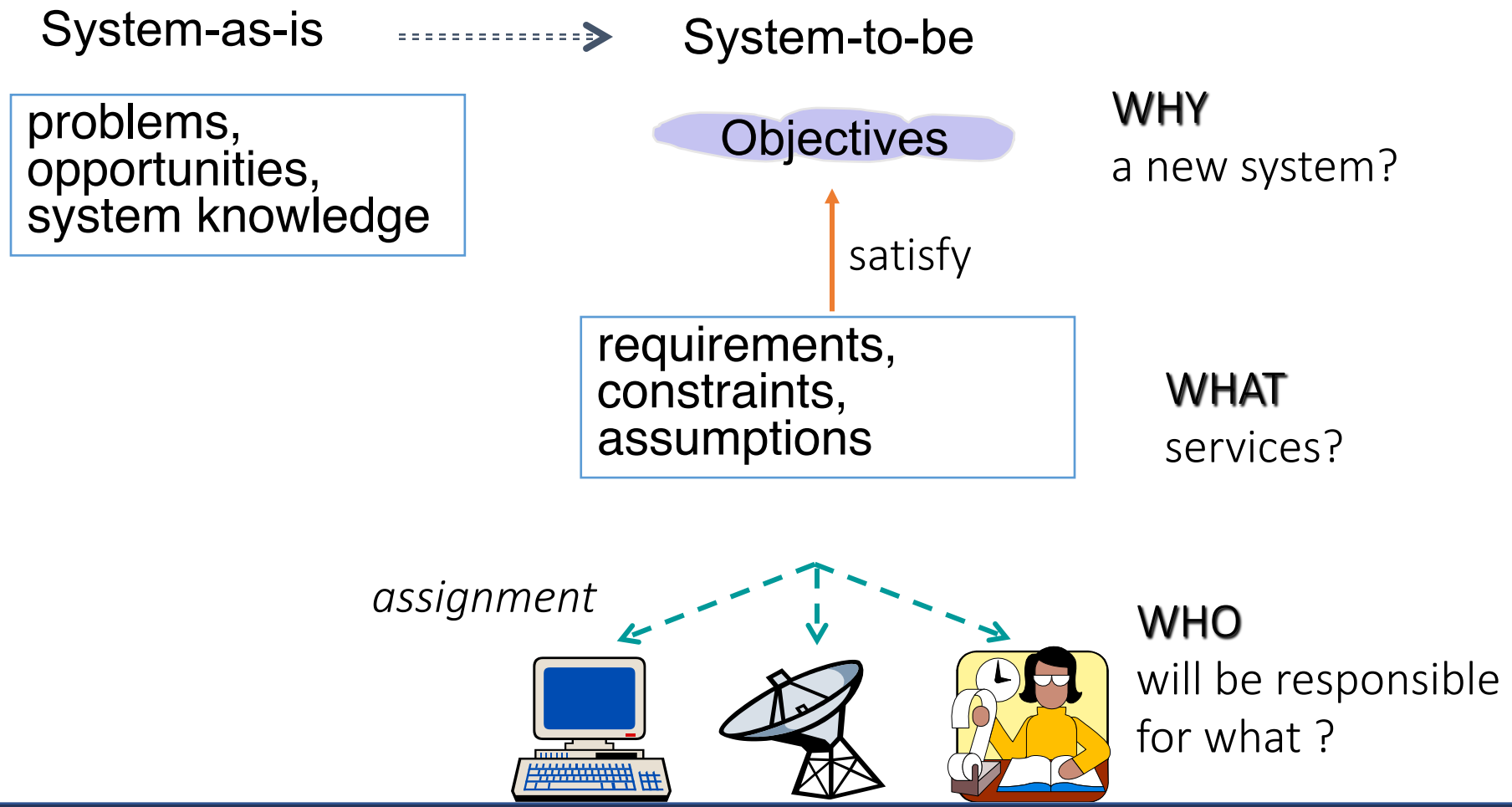


Examples

- **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.”

Activities of Requirements Engineering

Why, What, Who of RE



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

Documenting requirements

- Free unrestricted text
- Structured text
- Diagrams
- Formal specifications
- …More on this next week!

Further Reading

- Van Lamsweerde A. Requirements engineering: From system goals to UML models to software. John Wiley & Sons; 2009. Chapter 1
- What are requirements? Steve Easterbrook. 2004 http://www.cs.toronto.edu/~sme/CSC340F/readings/FoRE-chapter02-v7.pdf

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